EECE 5639 Computer Vision I

Lecture 19

Tracking: KLT, Camshift, Tracking as Classification, Circulant tracker

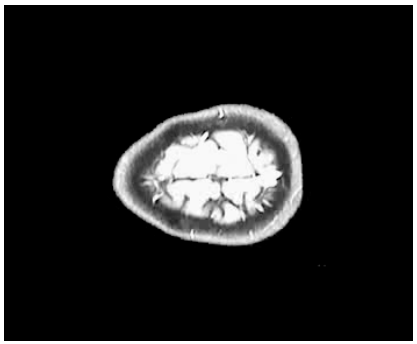
Project 4 is out. Due April 12.

Next Class

Dynamics-based Tracking

SFM

Tracking Applications



Template Matching & LK Tracking

Basic Template Matching

- Assumptions:
  - A snapshot of the target can be used to describe it
  - Target does not change (much) between frames
  - Motion is mostly translational

The last two assumptions are very restrictive.
Most current research works on relaxing these.

Template Matching

Is a search problem:

- Given a “template” from a previous frame, search for the corresponding region in the current frame
- Typically, use constraints to reduce the search space
- Need a “matching function” to evaluate similarity

Some Practical Issues

Object shape might not be well described by a patch aligned with the image axes.

We end including lots of the background in the template.

Some Practical Issues

A solution: use a Gaussian windowing function to weight pixels on the object more than pixels in the background

Works best with “compact” targets such as vehicles.

Some Practical Issues

Do not want to search the whole image for the target.
Solution: search a bounded region around likely position

Need: a prediction of where to look (more of this later).

Target appearance might change.
Solution: use an adaptive template

Need: a rule to change it (more of this later).

Lucas Kanade Tracking

Consider the SSD error function:

$$E(u,v) = \sum_{x,y} [I(x+u, y+v) - T(x,y)]^2$$

Shifted image

template

Score when template is at location (u,v)

Lucas Kanade Tracking

Consider the SSD error function:

$$E(u,v) = \sum_{x,y} [I(x+u, y+v) - T(x,y)]^2$$

Shifted image

template

Score when template is at location (u,v)

We want to find the location (u,v) where E is **minimum**.

Approach: steepest descent.

Lucas Kanade Tracking

$$E(u,v) = \sum_{x,y} [I(x+u, y+v) - T(x,y)]^2$$

Want to compute the gradient of this wrt u and v and set it to 0.

Steps:

1) Use a 1st order approximation of E
2) Take partial derivatives wrt u and v
3) Set them to 0
4) Solve a linear system of equations

First Order Approximation

$$
\begin{aligned}
E(u,v) &= \sum_{x,y} [I(x+u, y+v) - T(x,y)]^2 \\
&\approx \sum_{x,y} [I(x,y) + uI_x(x,y) + vI_y(x,y) - T(x,y)]^2 \\
&\approx \sum_{x,y} [uI_x(x,y) + vI_y(x,y) + D(x,y)]^2
\end{aligned}
$$

Taking Partial Derivatives

$$E(u,v) \approx \sum_{x,y} \left[uI_x(x,y) + vI_y(x,y) + D(x,y)\right]^2$$

$$\frac{\partial E}{\partial u} = 2\sum_{x,y} \left[uI_x(x,y) + vI_y(x,y) + D(x,y)\right] I_x(x,y)$$

$$\frac{\partial E}{\partial v} = 2\sum_{x,y} \left[uI_x(x,y) + vI_y(x,y) + D(x,y)\right] I_y(x,y)$$

Set them to 0 and solve for u and v

$$\sum_{x,y} \left[uI_x(x,y) + vI_y(x,y) + D(x,y) \right] I_x(x,y) = 0$$

$$\sum_{x,y} \left[uI_x(x,y) + vI_y(x,y) + D(x,y) \right] I_y(x,y) = 0$$

$$\sum_{x,y} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_u^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\sum_{x,y} \begin{bmatrix} I_x D \\ I_y D \end{bmatrix}$$

Again, this is the matrix used for corner detection!

Lucas Kanade Tracking

Traditional LK is typically run on small (5x5) corner like features.

Observation:
Can we use the same approach on larger windows, say a b.box around the target?

LK Tracking

In our previous derivation we made implicit assumptions …

1) brightness constancy assumption
2) all points in the template moved the same

For larger windows that might be a stretch






Fix by changing the motion parameters:

Some simple models: affine and projective.

Remember …

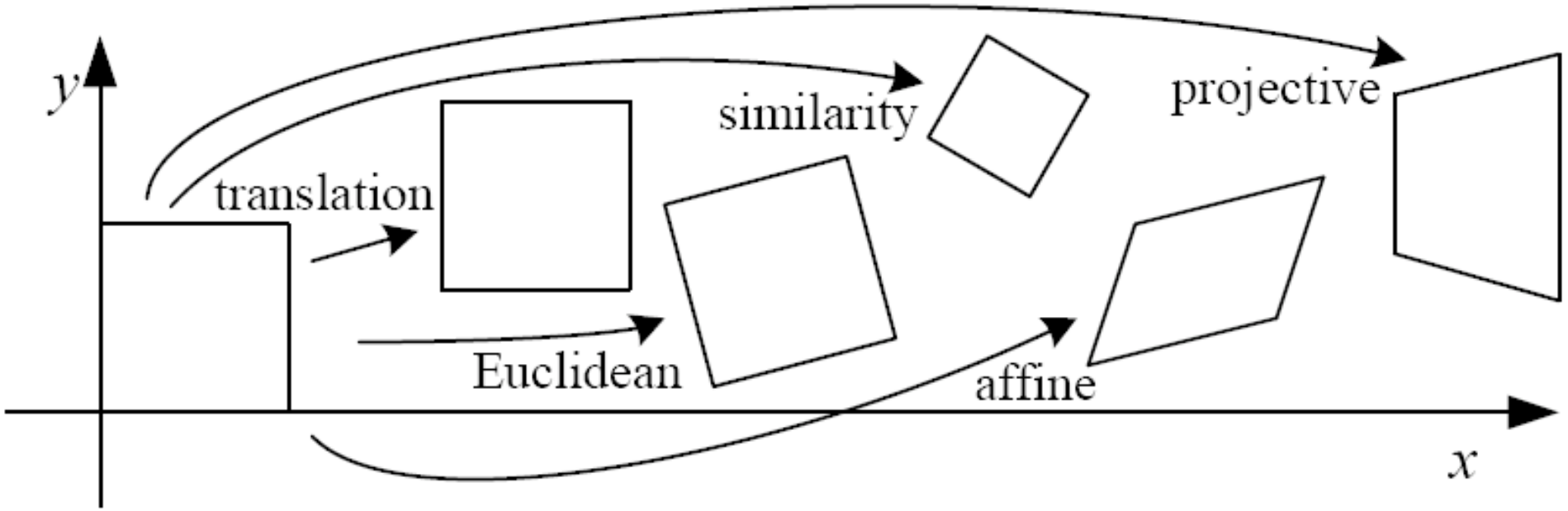
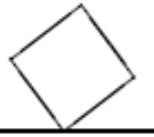
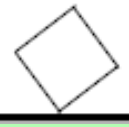
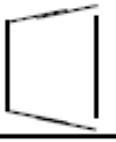


FIGURE 1. Basic set of 2D planar transformations

Summary of 2D Transformations

Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	2	orientation $+\cdots$	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	3	lengths $+\cdots$	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	4	angles $+\cdots$	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times 3}$	6	parallelism $+\cdots$	
projective	$\begin{bmatrix} \boldsymbol{H} \end{bmatrix}_{3\times 3}$	8	straight lines	

Back to LK Tracking …

Consider a warping function W with parameters P: W([x,y];P)

Examples:

1) Translation (u,v) (OF) P=[u,v]
2) Affine transformation P has 6 parameters (A,b)

Then, the objective is to minimize the matching score wrt P

$$E(u,v) = \sum_{x,y} [I((W(x,y);P)) - T(x,y)]^2$$

Chain Rule

$$z = f(x, y)$$ Image gray values

$$x = x(t)$$
$$y = y(t)$$ Warping of image coordinates

$$\frac{\partial f}{\partial t} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial t}$$

Taylor Expansion

$$I = I(x, y; p_1, p_2, \ldots, p_n)$$

After warping with slightly changed parameters:

$$I' = I(x, y; p_1 + \Delta p_1, p_2 + \Delta p_2, \ldots, p_n + \Delta p_n)$$

Using Taylor's expansion:

$$I' \approx I(x, y; p_1, p_2, \ldots, p_n) + \left[\frac{\partial I}{\partial x}\frac{\partial x}{\partial p_1} + \frac{\partial I}{\partial y}\frac{\partial y}{\partial p_1}\right]\Delta p_1 + \left[\frac{\partial I}{\partial x}\frac{\partial x}{\partial p_2} + \frac{\partial I}{\partial y}\frac{\partial y}{\partial p_2}\right]\Delta p_2 + \ldots + \left[\frac{\partial I}{\partial x}\frac{\partial x}{\partial p_n} + \frac{\partial I}{\partial y}\frac{\partial y}{\partial p_n}\right]\Delta p_n$$

Image Gradient

Warp Jacobian

LK Tracking

Use steepest descent to find (local) min.

$$E(P) = \sum_{x,y} \left[I((W(x,y);P)) - T(x,y) \right]^2$$

$$E(P + \Delta P) \approx \sum_{x,y} \left[I((W(x,y);P)) + \nabla I \frac{\partial W}{\partial P} \Delta P - T(x,y) \right]^2$$

Image gradient

Warp Jacobian

Example: Affine warping (6 parameters)

$$W = \begin{bmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Warp Jacobian

$$W([x, y]; P) = (W_x, W_y)$$

$$\frac{\partial W}{\partial P} = \begin{bmatrix} \frac{\partial W_x}{\partial p_1} & \frac{\partial W_x}{\partial p_2} & \cdots & \frac{\partial W_x}{\partial p_n} \\ \frac{\partial W_y}{\partial p_1} & \frac{\partial W_y}{\partial p_2} & \cdots & \frac{\partial W_y}{\partial p_n} \end{bmatrix}$$

Warp Jacobian, Affine warping

$$W = \begin{bmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{aligned} W_x &= (1+p_1)x + p_3 y + p_5 \\ W_y &= p_2 x + (1+p_4)y + p_6 \end{aligned}$$

Warp Jacobian, Affine warping

$$
\begin{aligned}
W_x &= (1+p_1)x + p_3 y + p_5 \\
W_y &= p_2 x + (1+p_4)y + p_6
\end{aligned}
$$

$$
\frac{\partial W}{\partial P} = \begin{bmatrix} \frac{\partial W_x}{\partial p_1} & \frac{\partial W_x}{\partial p_2} & \cdots & \frac{\partial W_x}{\partial p_n} \\ \frac{\partial W_y}{\partial p_1} & \frac{\partial W_y}{\partial p_2} & \cdots & \frac{\partial W_y}{\partial p_n} \end{bmatrix}
$$

$$
\frac{\partial W}{\partial P} = \begin{bmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{bmatrix}
$$

Taking derivatives of E wrt P and setting them to 0:

$$E(P+\Delta P) \approx \sum_{x,y} \left[I((W(x,y);P)) + \nabla I \frac{\partial W}{\partial P} \Delta P - T(x,y) \right]^2$$

$$\sum_{x,y} \left\{ [\nabla I \frac{\partial W}{\partial P}]^T [(I(W[x,y];P)) + \nabla I \frac{\partial W}{\partial P} \Delta P - T(x,y)] \right\} = 0$$

- Solve for Delta P;
- Update P: P = P + Delta P
- Iterate until Delta P is negligible

Solve for Delta P:

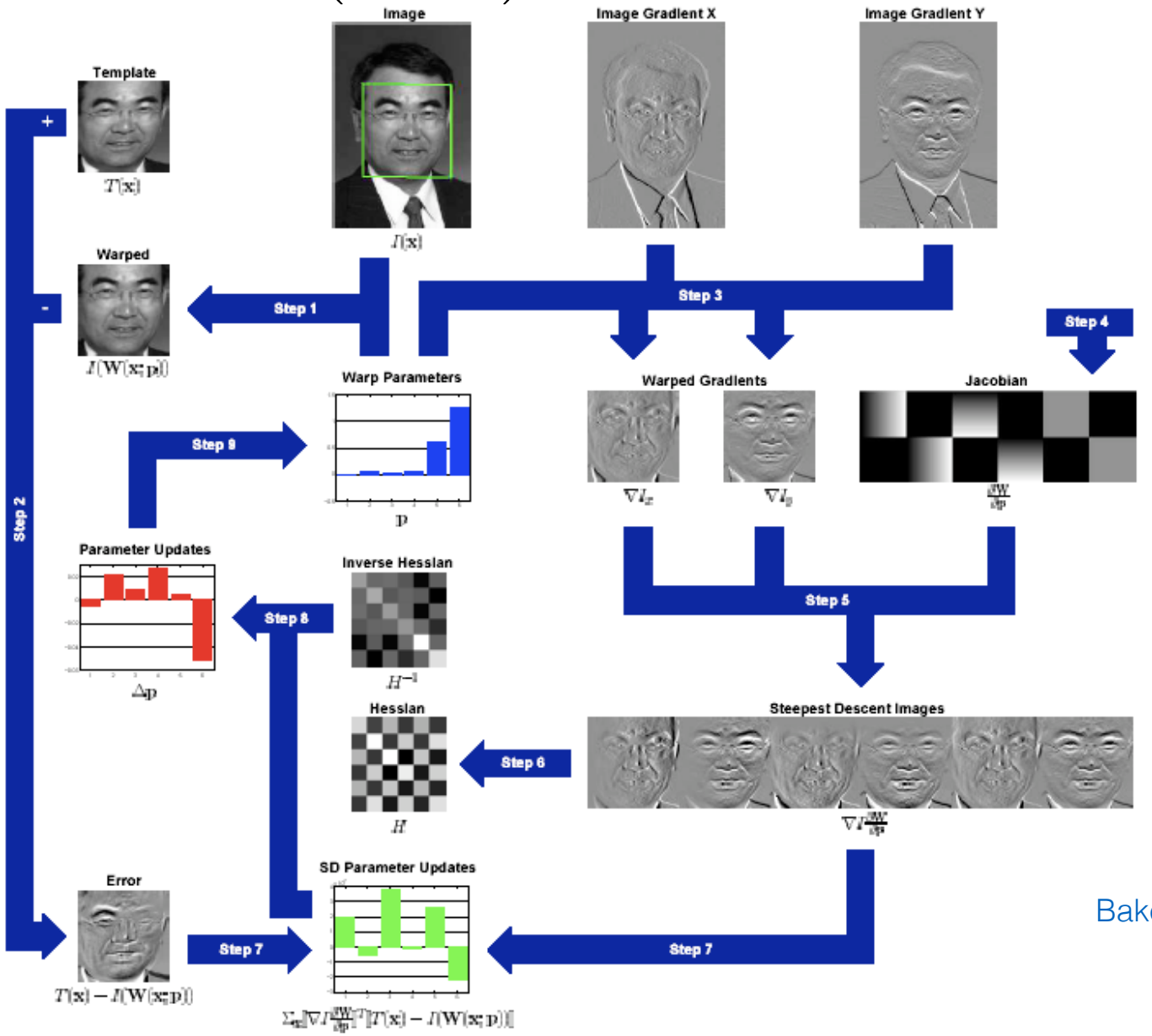
$$\sum_{x,y} \left\{ [\nabla I \frac{\partial W}{\partial P}]^T [(I(W[x,y];P)) + \nabla I \frac{\partial W}{\partial P} \Delta P - T(x,y)] \right\} = 0$$

Gauss-Newton Approximation of the Hessian

$$\begin{aligned} \Delta P &= [\sum (\nabla I \frac{\partial W}{\partial P})^T \nabla I \frac{\partial W}{\partial P}]^{-1} \sum (\nabla I \frac{\partial W}{\partial P})^T T(x,y) - (I(W[x,y];P)))] \\ &= H^{-1} \sum (\nabla I \frac{\partial W}{\partial P})^T T(x,y) - (I(W[x,y];P)))] \end{aligned}$$

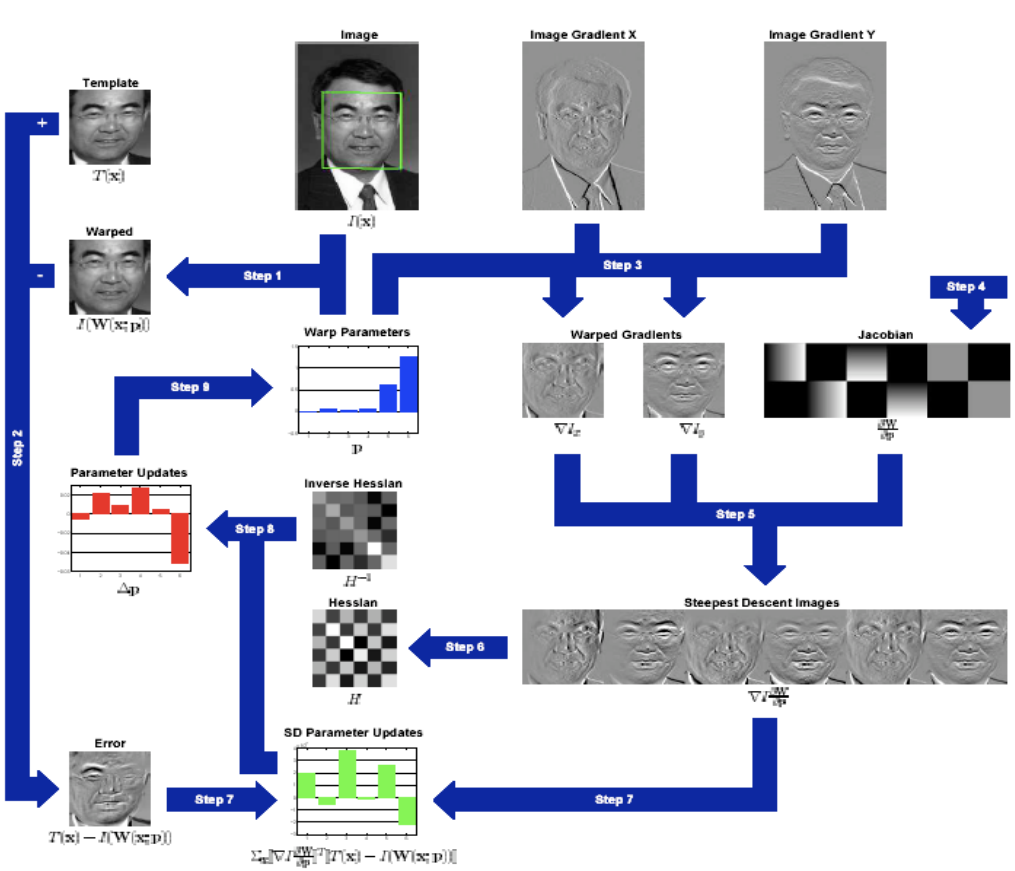
Algorithm at a glance

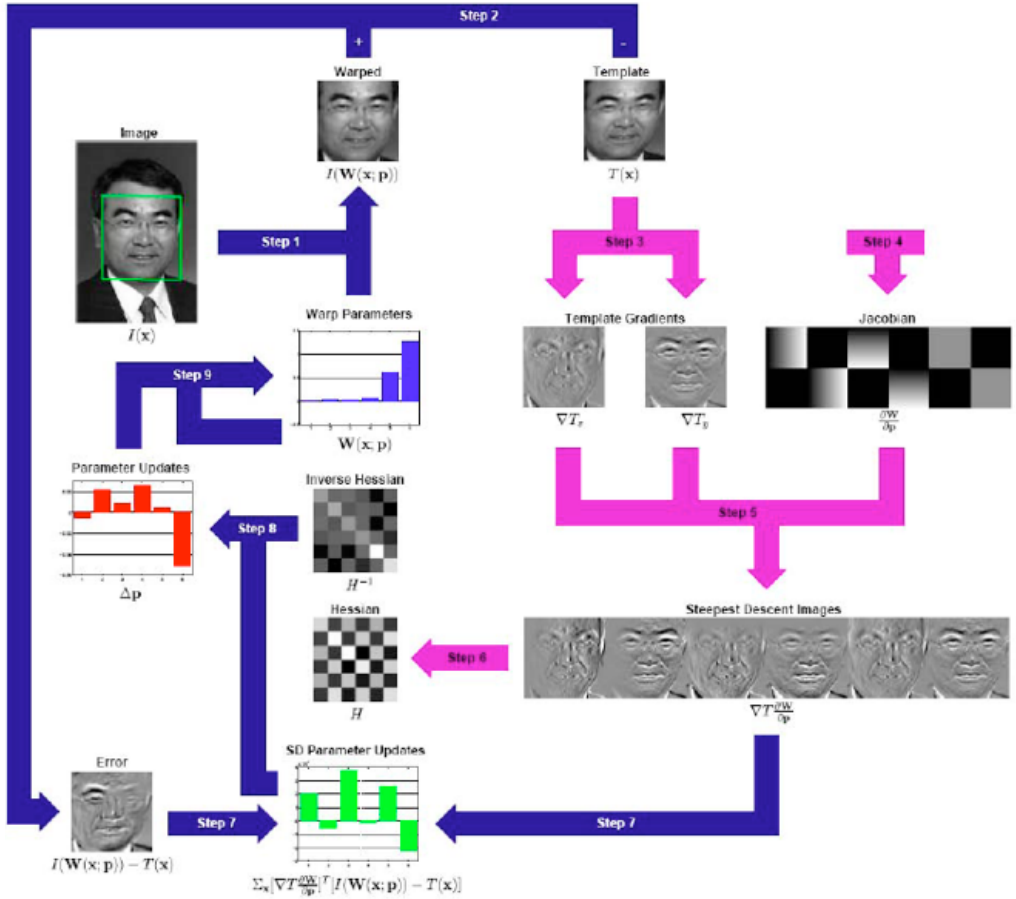
$$\Delta P = H^{-1} \sum \left(\nabla I \frac{\partial W}{\partial P} \right)^T [T(x,y) - I(W[x,y];P)]$$



Baker & Matthews
IJCV 04

Inverse Compositional Variant





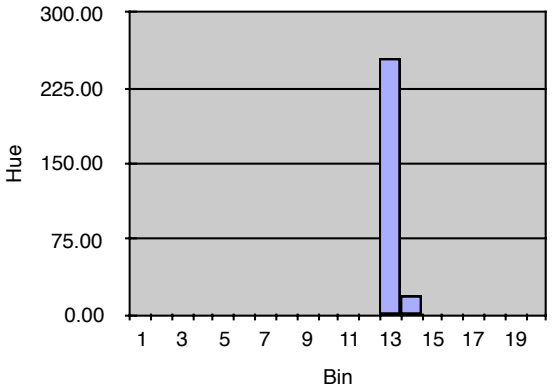
Brightness Changes

Traditional LK assumes brightness constancy.

We can remove potential error sources by first normalizing the template and the local search window, by taking off the mean and dividing by the std. dev.

Color-based tracking

1.Set initial search window



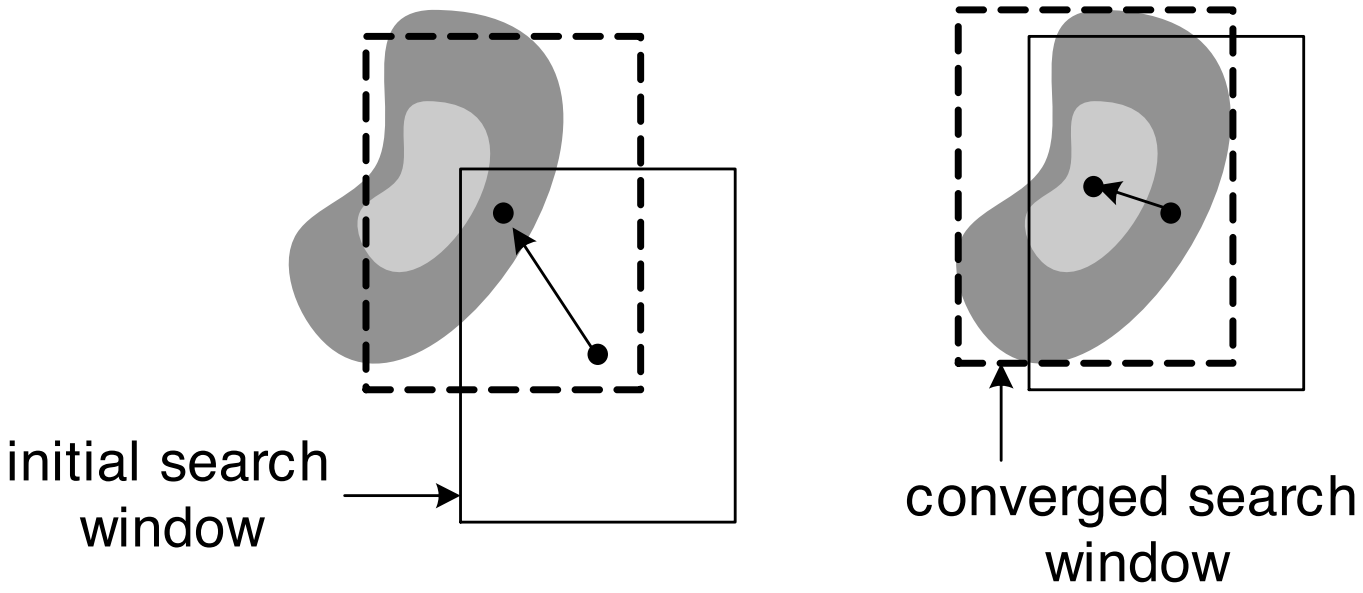
2.Calculate histogram

3.Original image

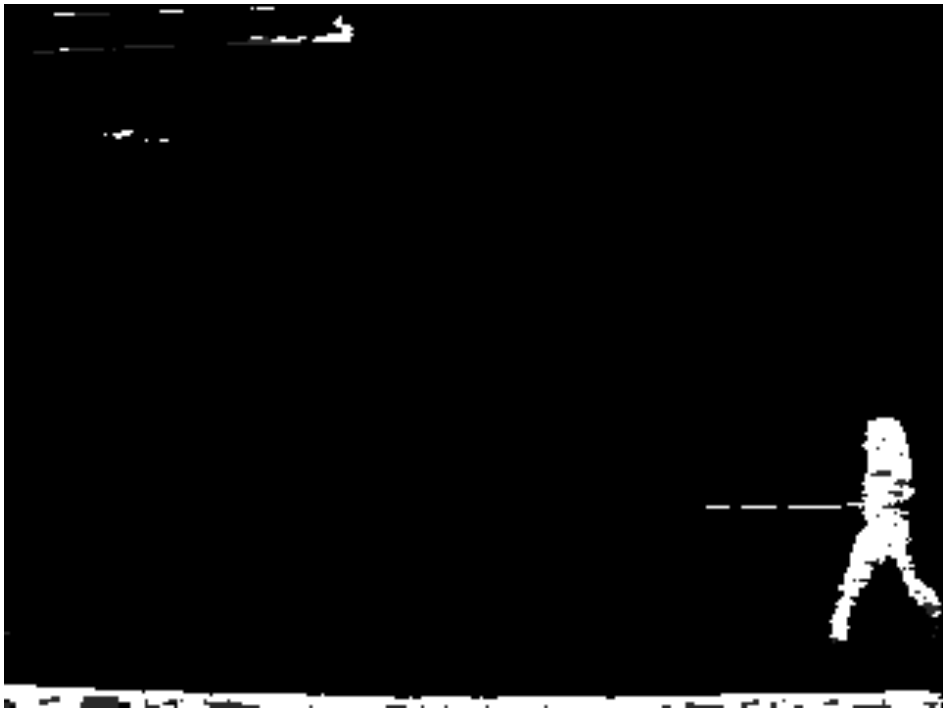
4.Probability image

Finding the most likely blob:

Mean Shift Algorithm: steepest ascent search of the mode:



Camshift Tracking



Tracking Colored Blobs:Algorithm CAMSHIFT

(Continuously Adaptive Mean Shift)

Application:

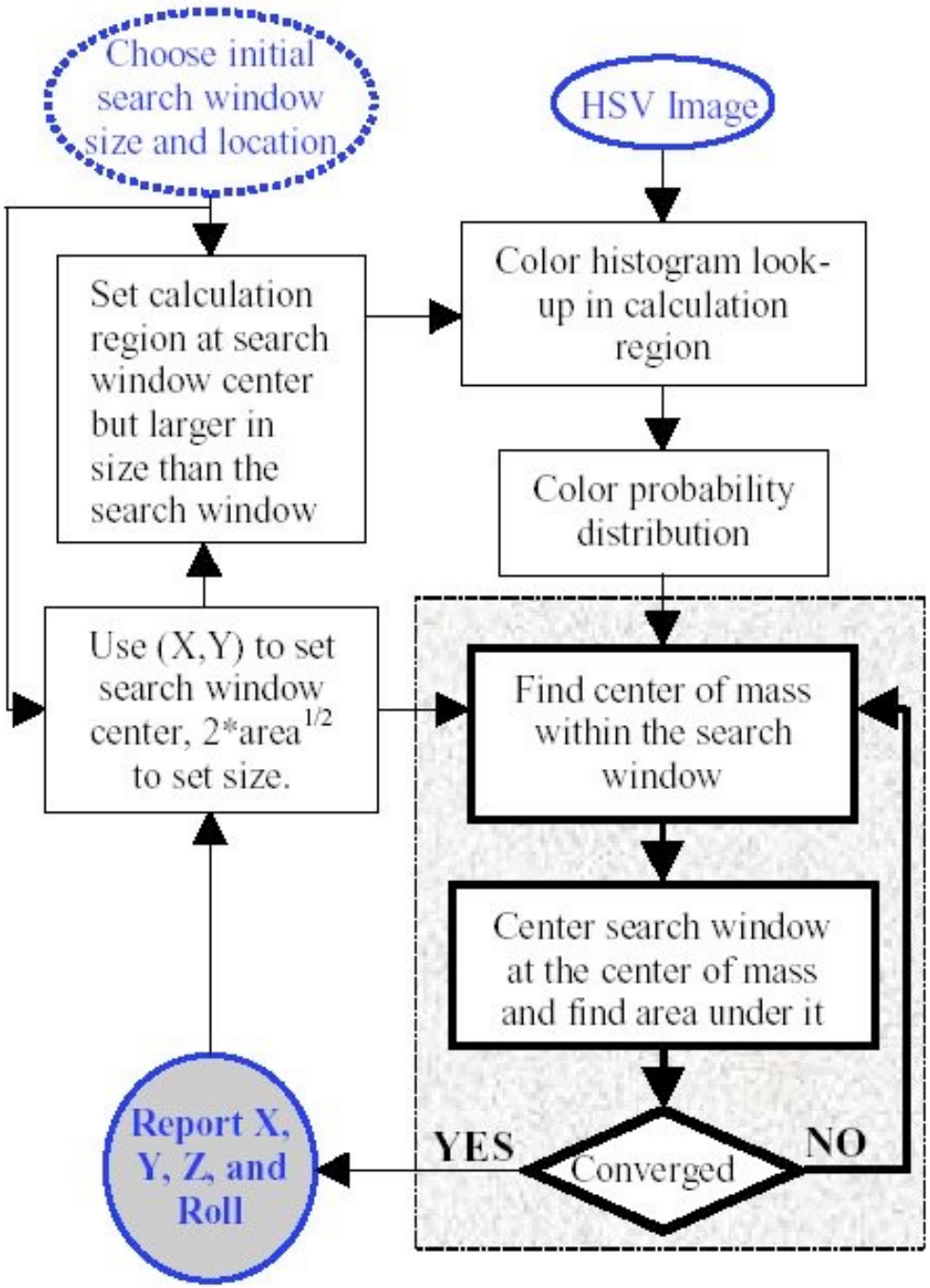
Tracking faces for hci, teleconferencing, etc

Approach:

Find pixels that have the same color distribution across frames by looking at their histogram in HSV color space

Based on “Mean Shift” algorithm

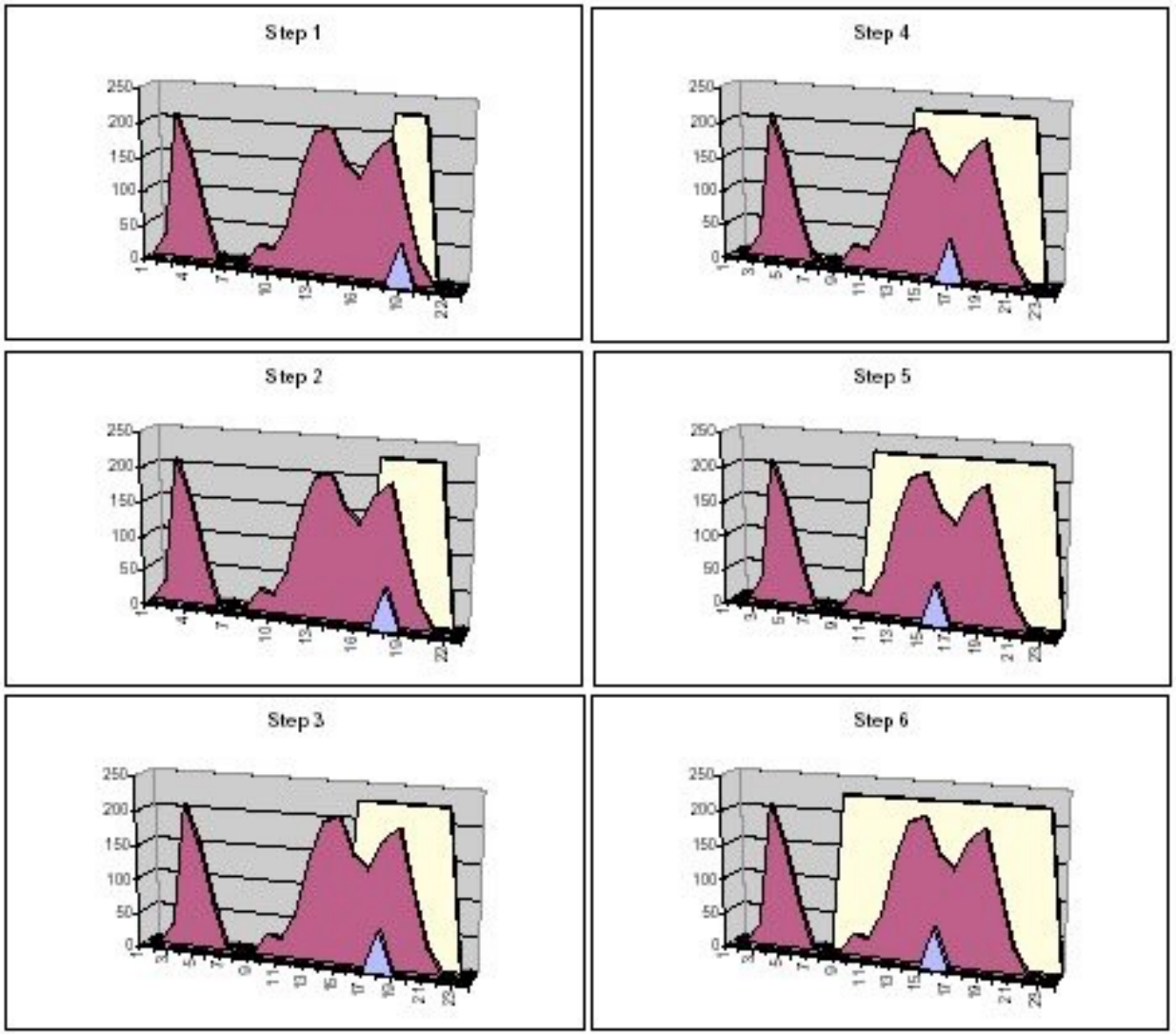
Adapts the size of the window at each frame


Choose initial search window size and location
HSV Image
Set calculation region at search window center but larger in size than the search window
Color histogram look-up in calculation region
Color probability distribution
Use (X,Y) to set search window center, 2*area^1/2 to set size.
Find center of mass within the search window
Center search window at the center of mass and find area under it
Converged
YES
NO
Report X, Y, Z, and Roll

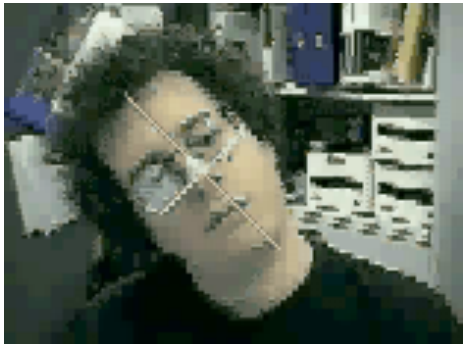
Algorithm CAMSHIFT

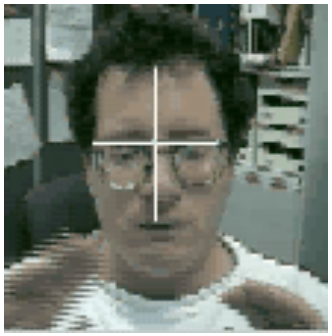
1. Choose initial window
2. Mean Shift (one or more iterations)
3. Record zeroth moment of the window $M_{oo} = \sum I(x,y)$
4. Set window size $s = 2 * (M_{oo}/256)^{1/2}$
5. Repeat steps 2 to 4 until convergence.

Camshaft Windows Iterations



EXAMPLES





Tracking as Classification

Online Adaptation

Goal: long-term tracking

The more invariant is the appearance model to variations in lighting and geometry, the less specific it is in representing a particular object.

Then, there is danger of getting confused by surroundings

Online adaptation allows us to retain specificity at each time frame while maintaining some generality about appearance changes .

Tracking as Classification (Collins & Liu '03)

"Online Selection of Discriminative Tracking Features" (iccv 03)

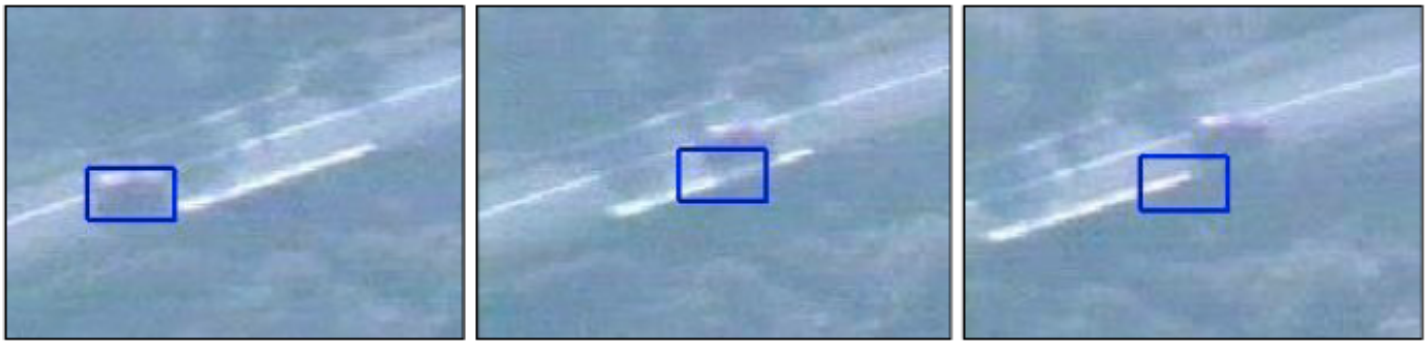
Target tracking posed as a binary classification problem:

discriminate between foreground (target) and background

This point of view brings in a wide range of classification techniques to the tracking problem.

Motivation

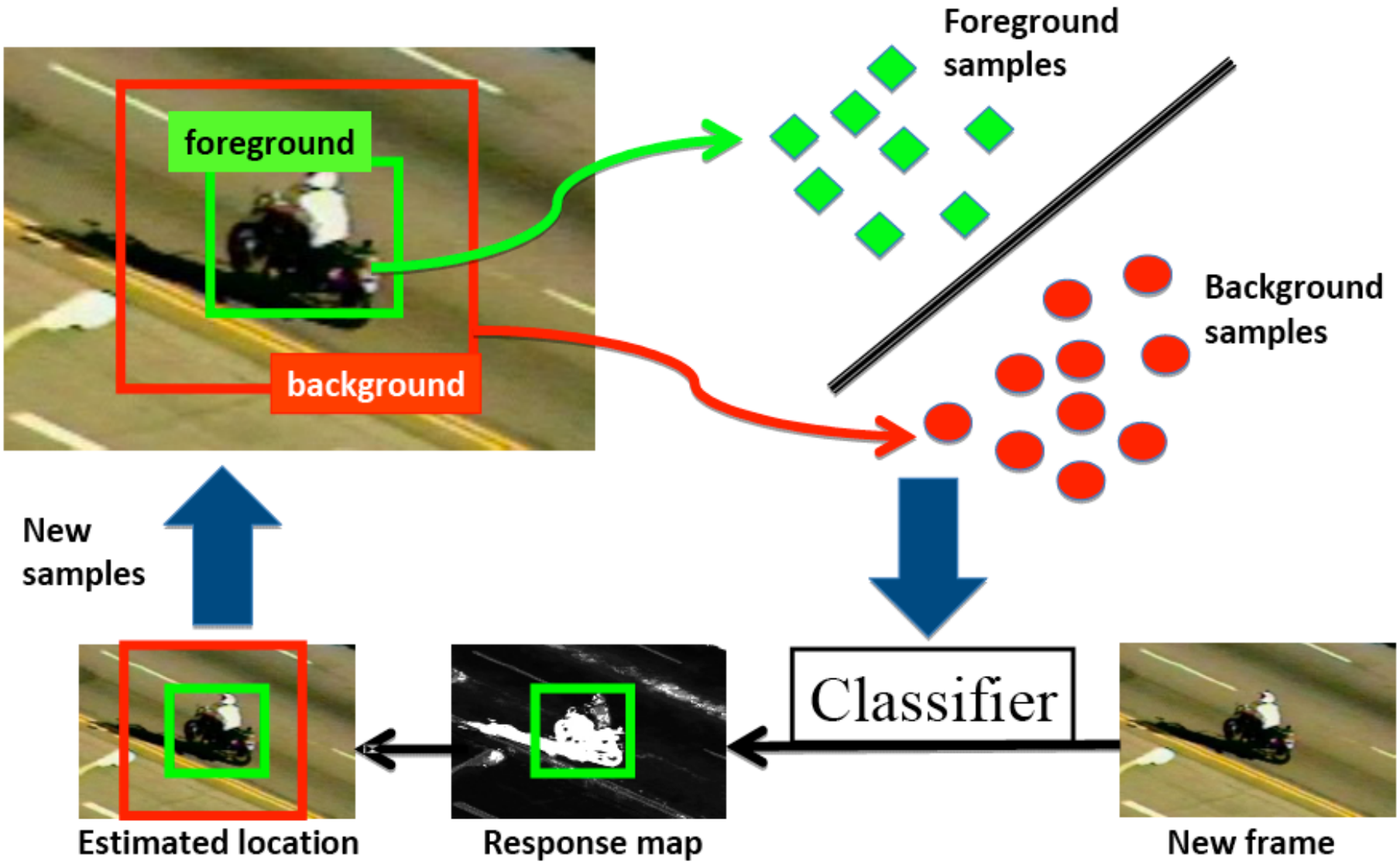
Tracking success/failure is highly correlated with our ability to distinguish the target appearance from the background (think camouflage)



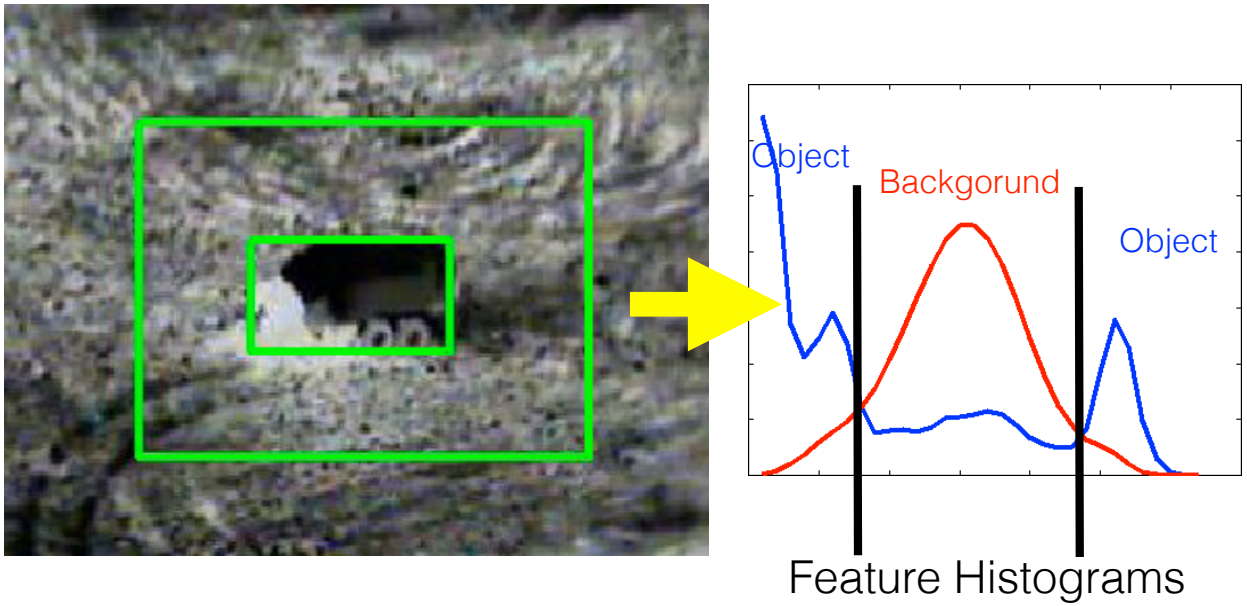
Suggestions:

- Explicitly seek “good” features that discriminate
- Continuously adapt the features

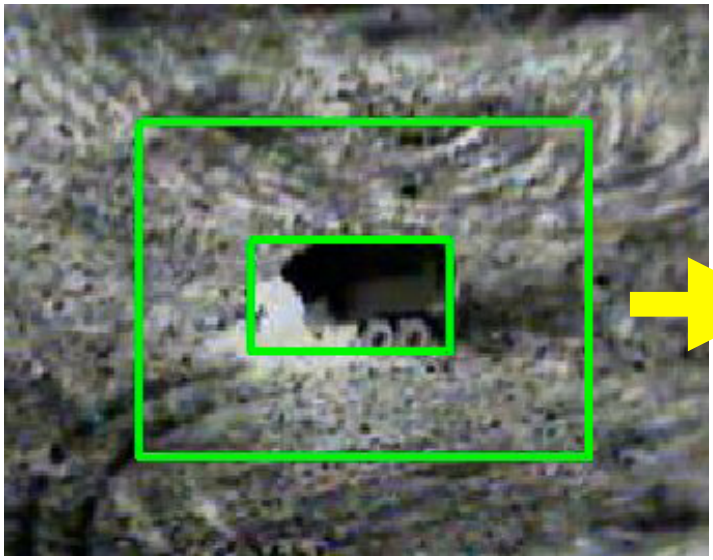
Overview


foreground
background
Foreground samples
Background samples
New samples
Classifier
Estimated location
Response map
New frame

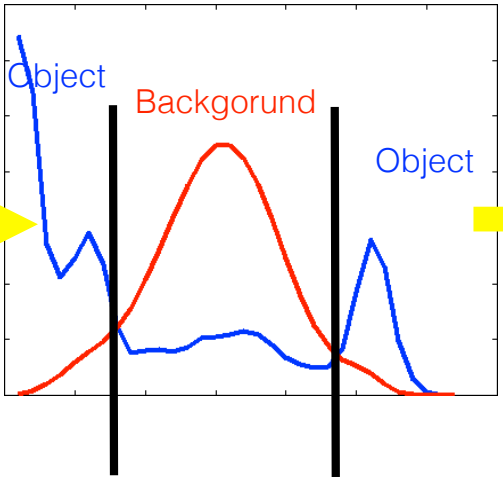
Evaluation of Feature Discriminant Power



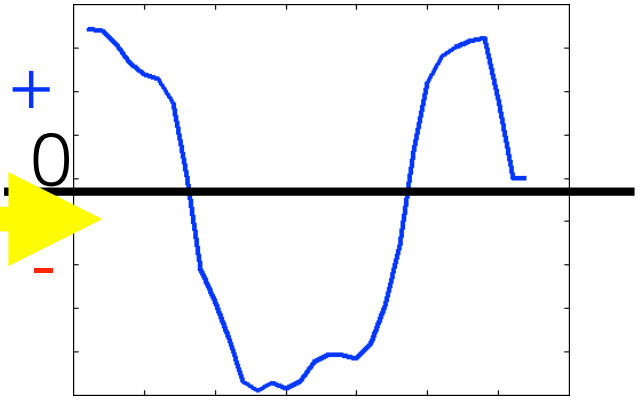
Evaluation of Feature Discriminant Power



Feature Histograms

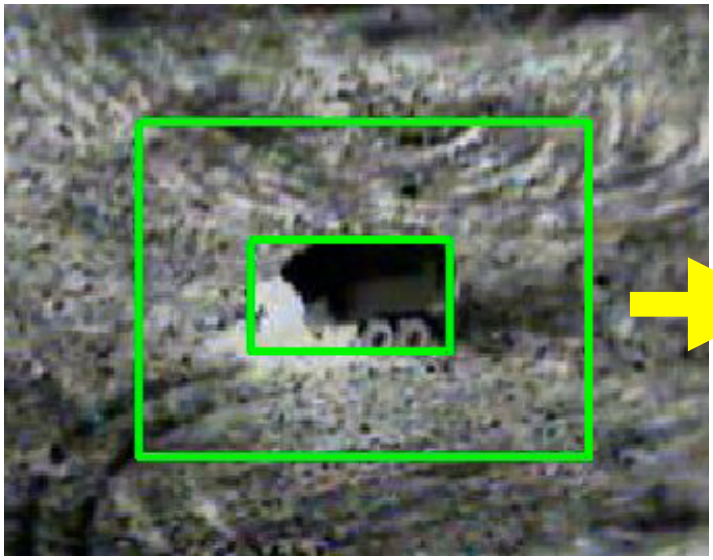


Log Likelihood Ratio

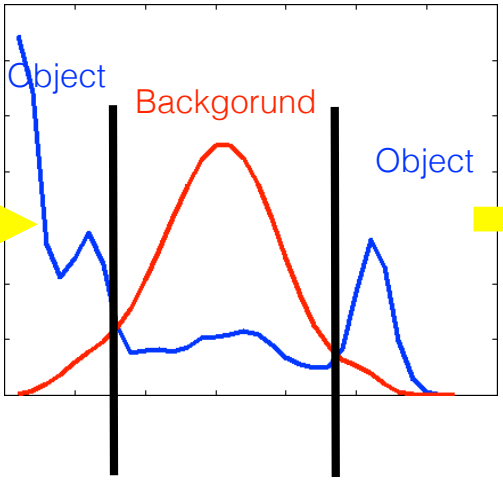


$$L(i) = log \frac{\max\{p(i), \delta\}}{\max\{q(i), \delta\}}$$

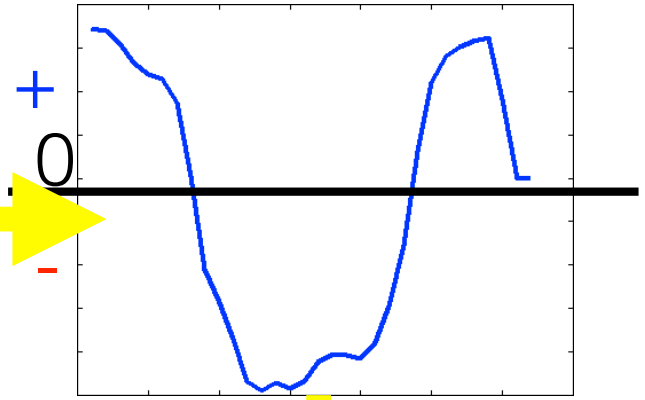
Evaluation of Feature Discriminant Power

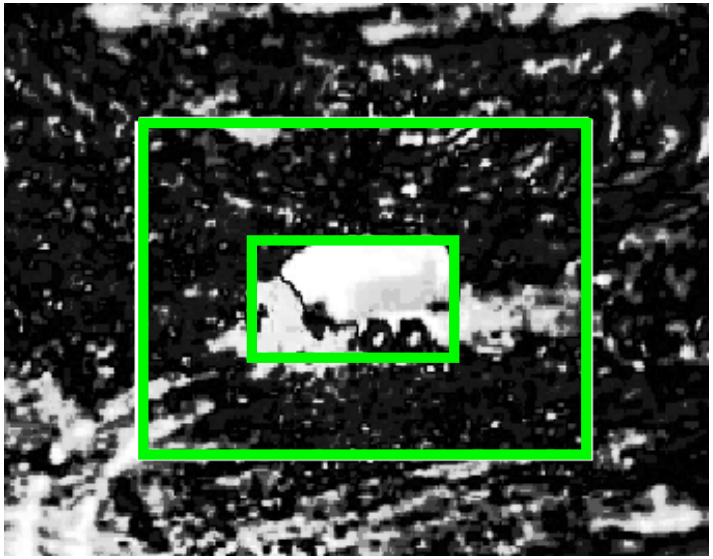


Feature Histograms



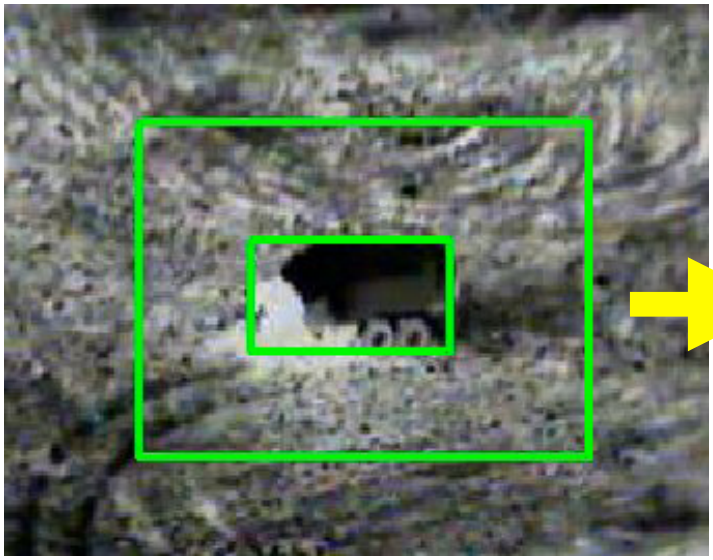
Log Likelihood Ratio



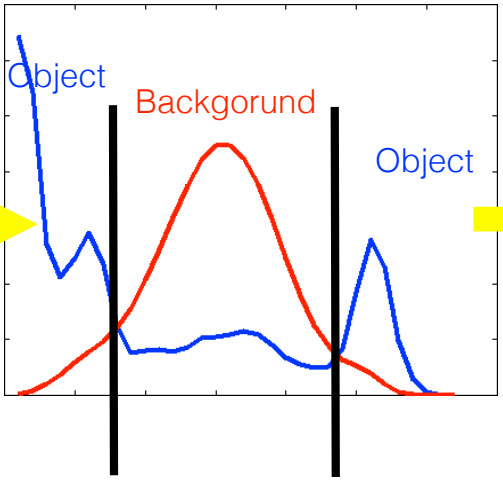


Likelihood Image

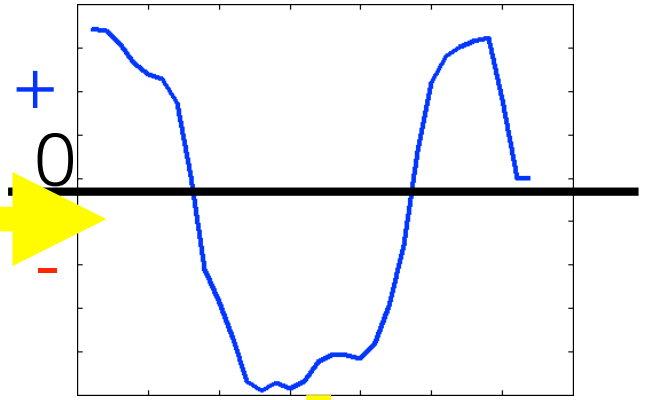
Evaluation of Feature Discriminant Power



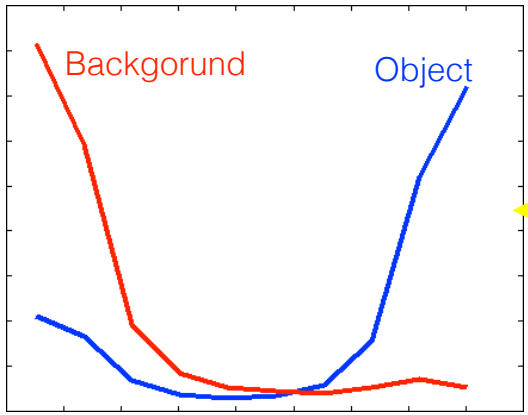
Feature Histograms

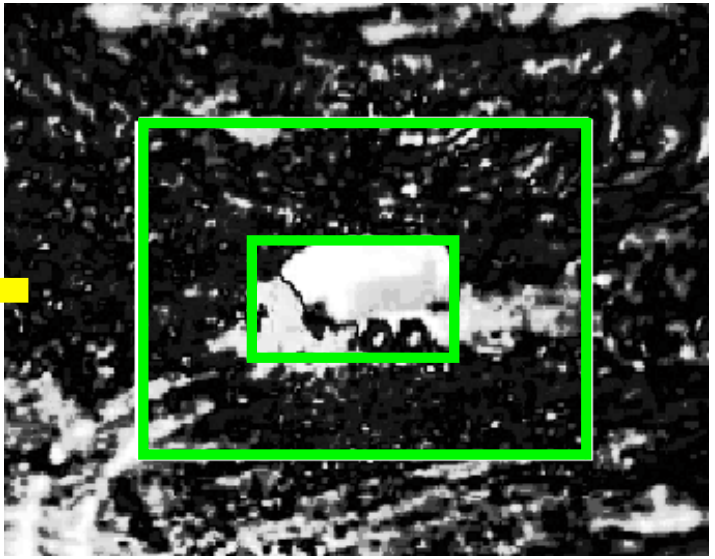


Log Likelihood Ratio



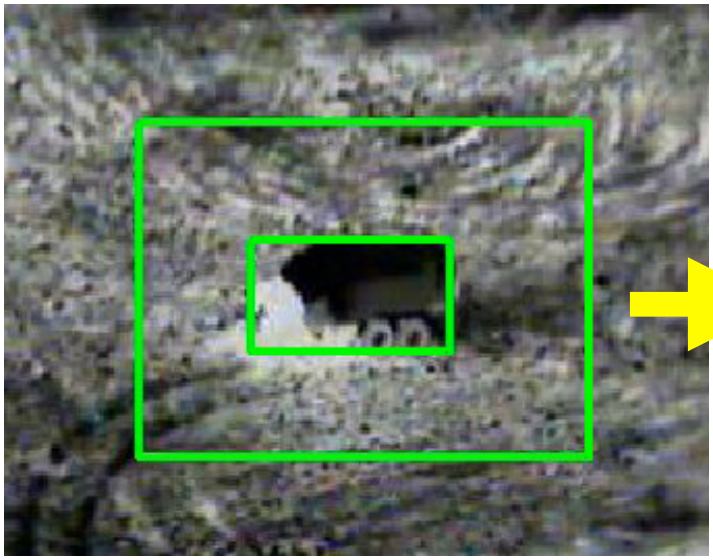
Likelihood Histograms



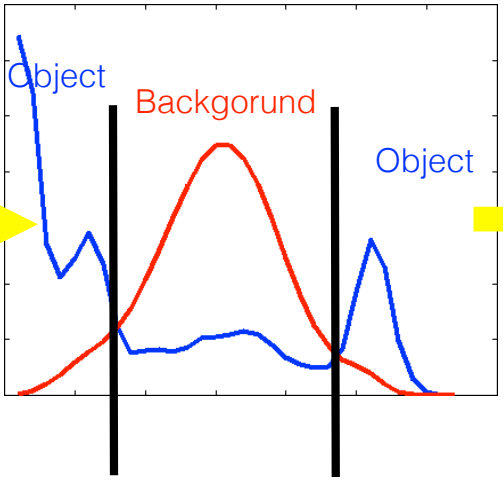


Likelihood Image

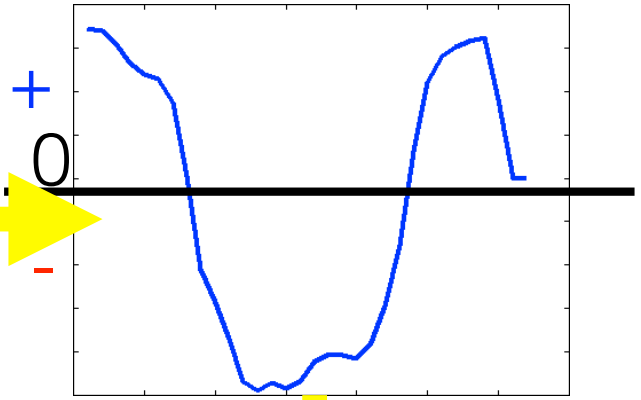
Evaluation of Feature Discriminant Power

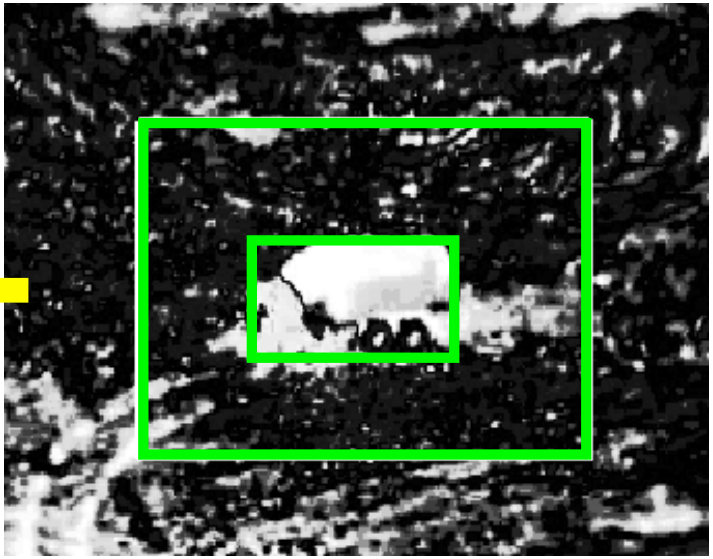


Feature Histograms



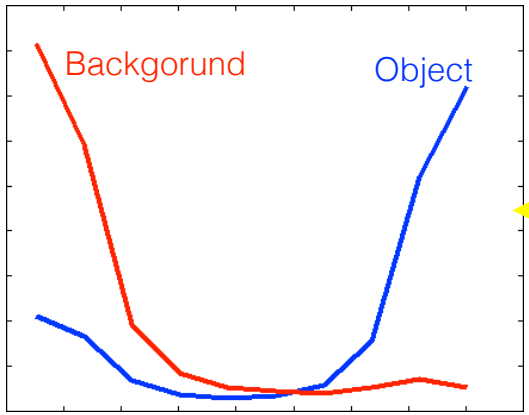
Log Likelihood Ratio





Likelihood Image

Likelihood Histograms



Feature Score

$$\frac{\text{Var between classes}}{\text{Var within classes}}$$

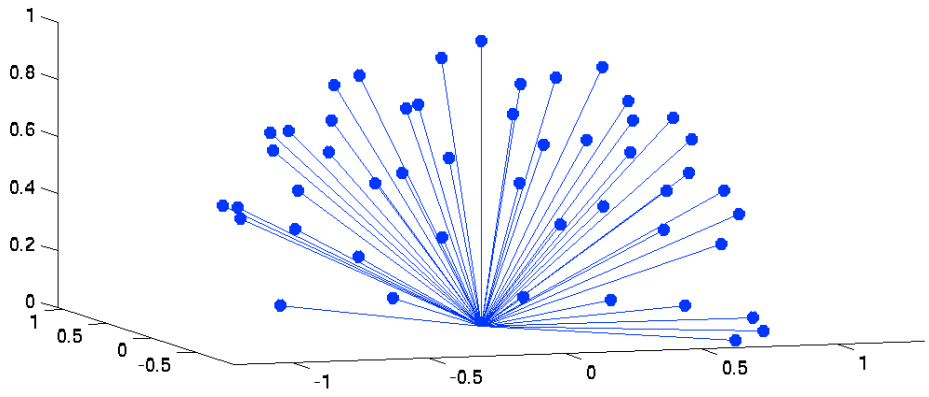
$$\mathrm{VR}(L;p,q) \equiv \frac{\mathrm{var}(L;(p+q)/2)}{[\mathrm{var}(L;p)+\mathrm{var}(L,q)]}$$

Example: 1D Color Feature Spaces

Color features: integer linear combinations of R,G,B

$$\frac{aR + bG + cB}{|a| + |b| + |c|} + \text{offset} \qquad a, b, c \in \{-2, -1, 0, 1, 2\}$$

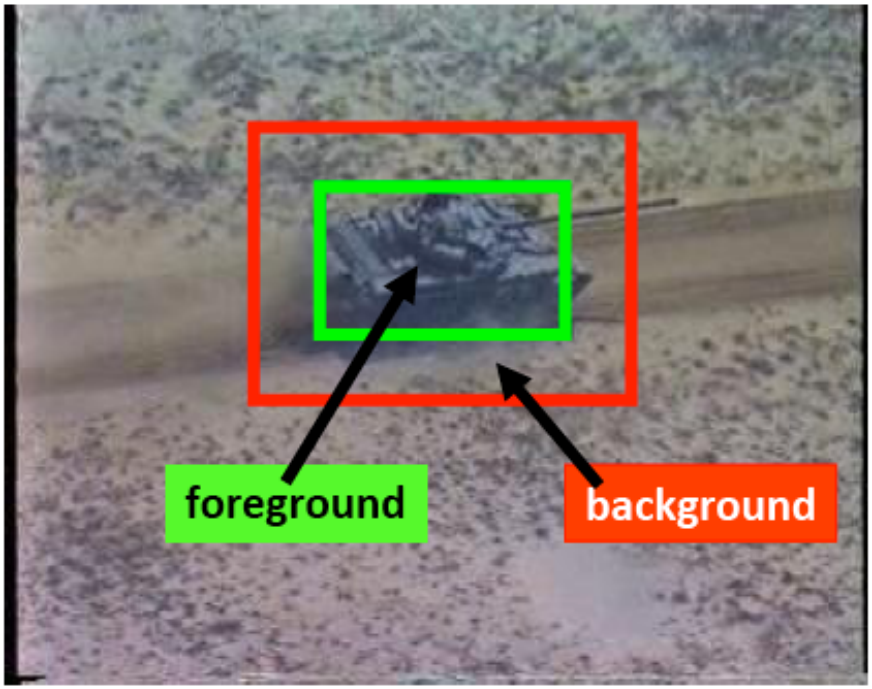
offset to bring feature back to 0-255 range



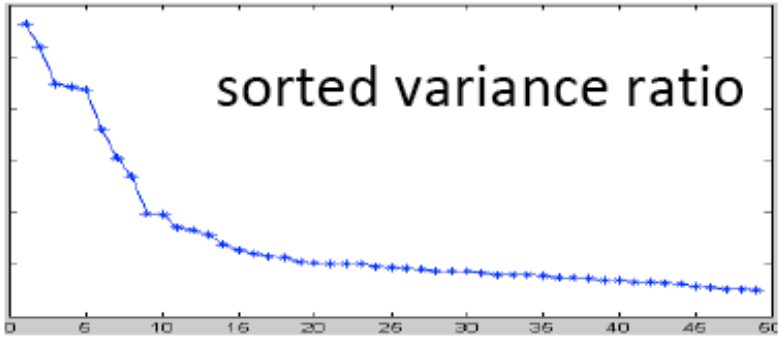
Once you remove duplicates and 0, there are 49 distinct features.
They roughly uniformly sample the 1D marginal distributions of RGB.

Example

training frame

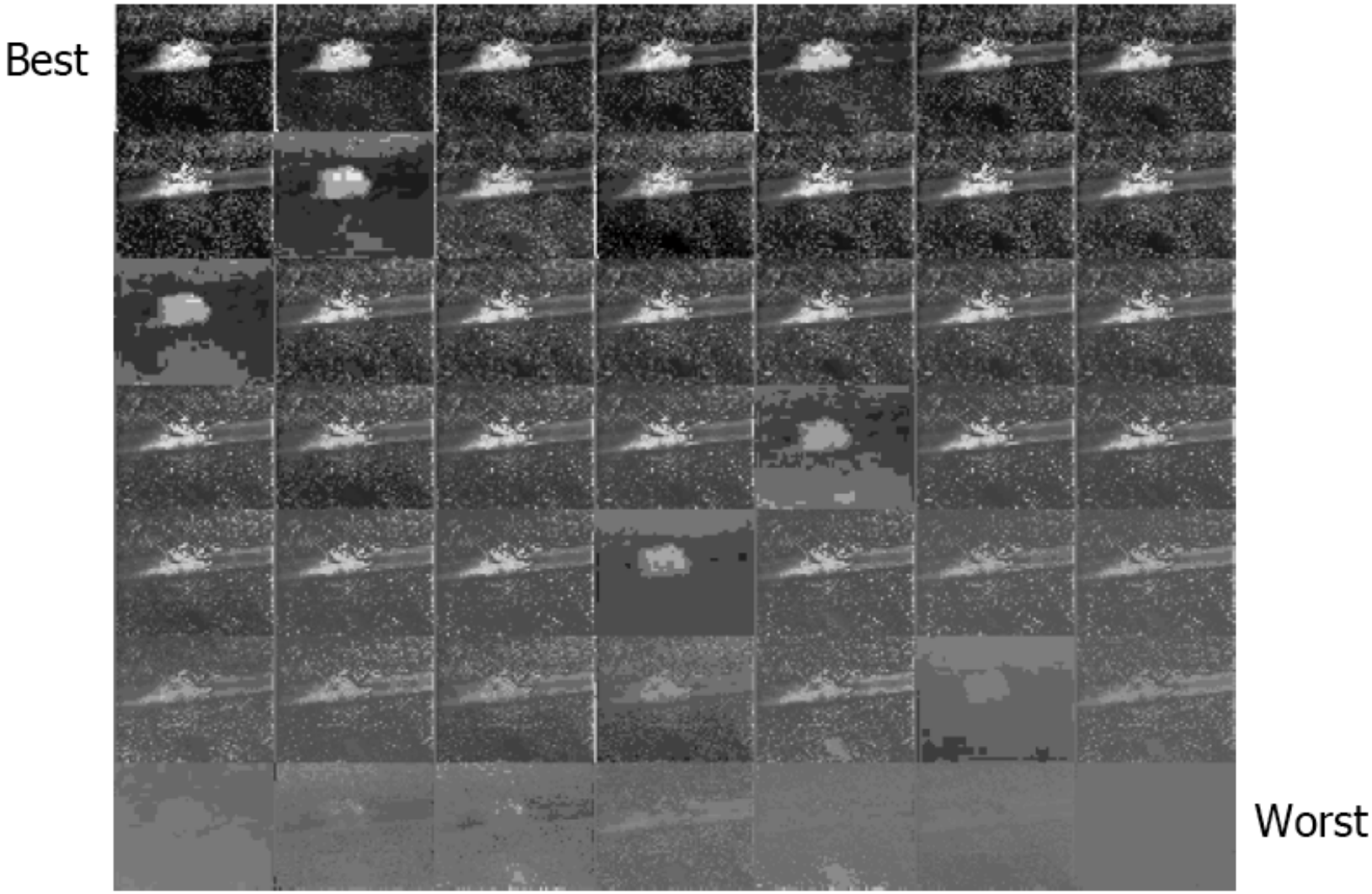


test frame

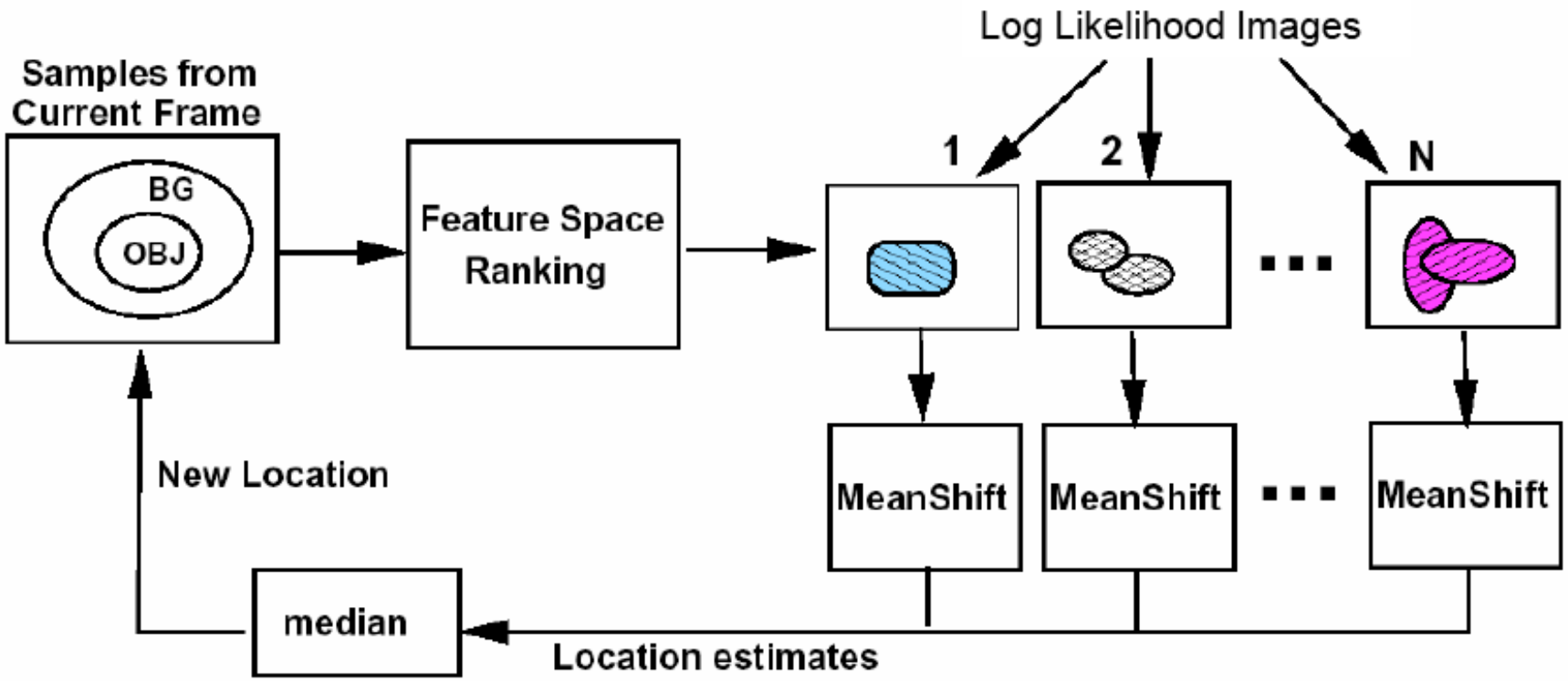


color features

Example: Feature Ranking



Overview Algorithm



Note: since log likelihood images contain negative values, must use modified mean-shift algorithm as described in Collins, CVPR'03

$$\Delta_x = \frac{\sum_a K(a-x)w(a)(a-x)}{\sum_a |K(a-x)w(a)|}$$

use absolute values

Avoiding Model Drift

The boundingbox on the target is not tight:

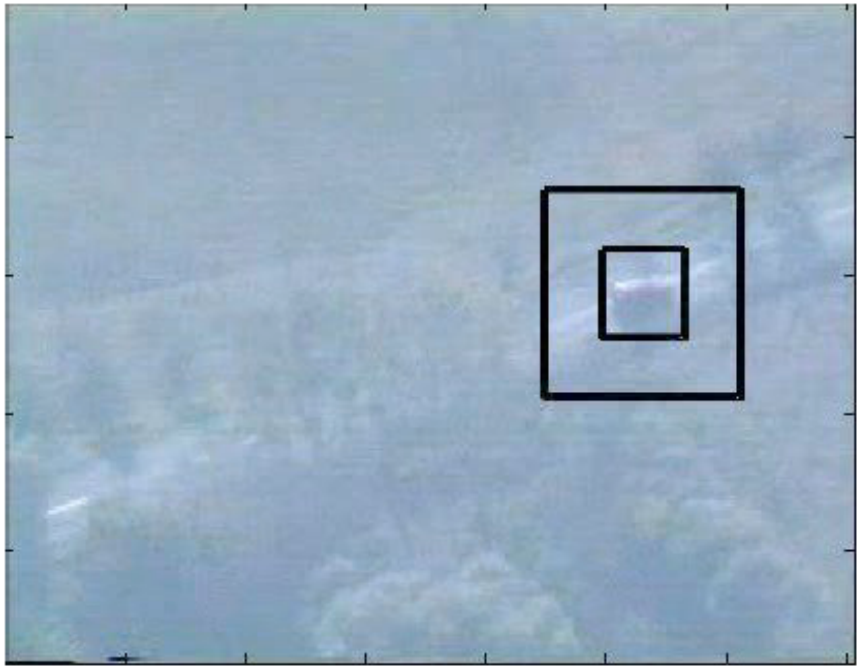
- we incorporate background pixels into the model
- we will have drift

Use anchoring:

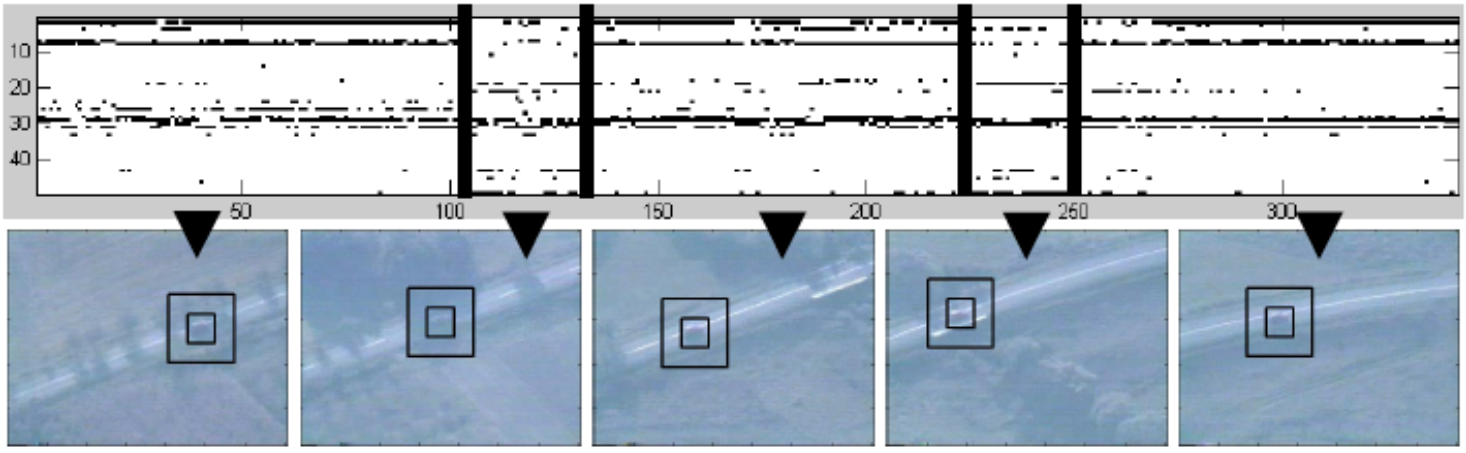
- combine current model with model from first frame
- solves drift, but limits the ability of adapting to new appearances

anchor distribution = object appearance histogram from first frame
model distribution = (current distribution + anchor distribution) / 2

Example: Hard-to-see Objects

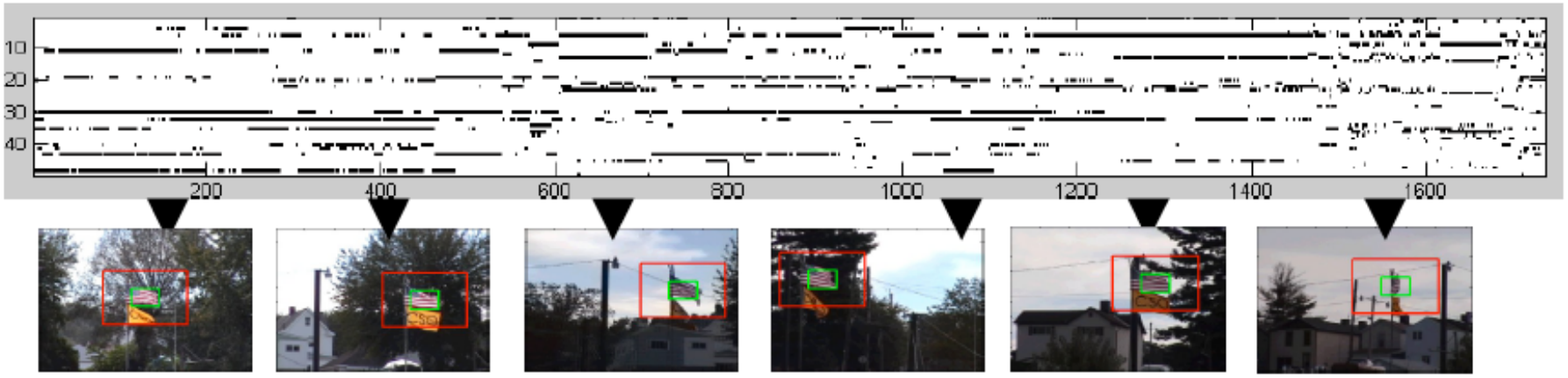


Trace of selected features



Example: Changing Illumination/Background

Trace of selected features



Tracking by Detection Using Dense Sampling

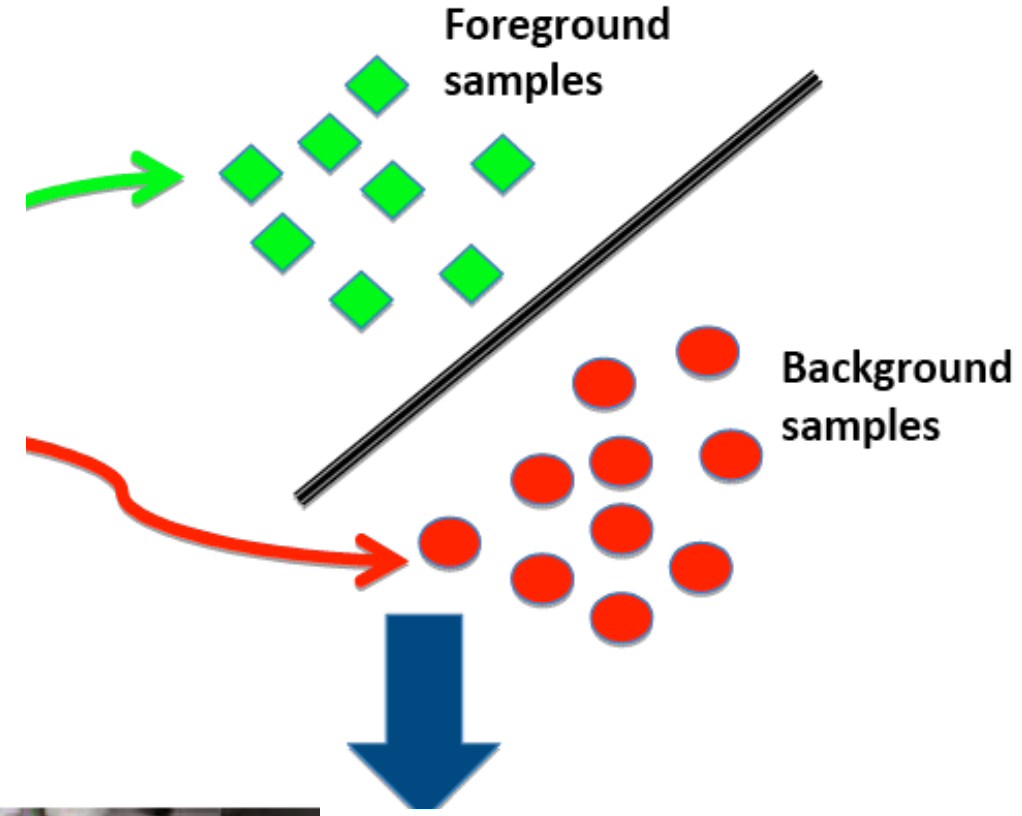
(Henriques, Caseiro, Martins & Batista, '12)

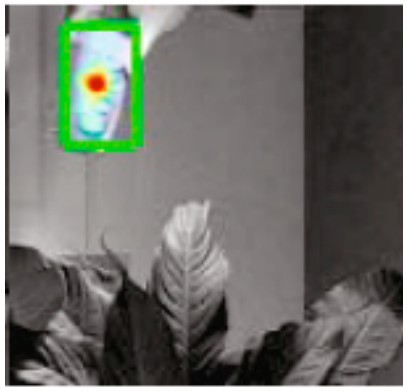
At each frame, collect a set of dense samples around the estimated location of the target.

Samples are continuously labeled (between 1 and 0) using a Gaussian centered at the target location with stdev s

Uses a Kernel Regularized Least Square Classifier

Overview





Classifier Design Using Labeled Data

Given labeled training samples:

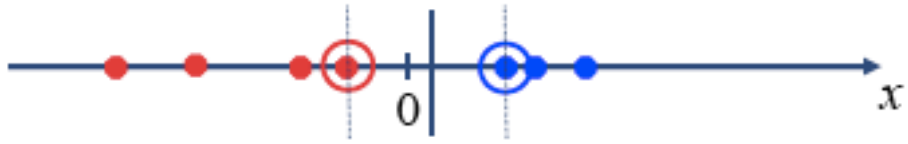
$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)$$

$$y_i \in \{-1, 1\}$$

- Would like to find a classifier function F(x; c) that given a sample x, would return its label.

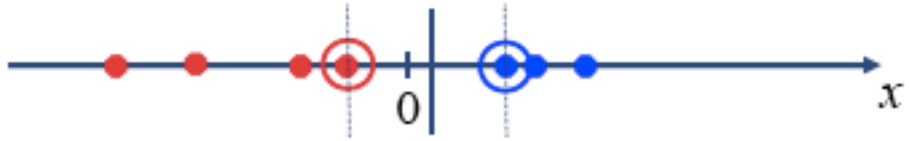
Classifier Using Labeled Data

- Datasets that are linearly separable work out great:



Classifier Using Labeled Data

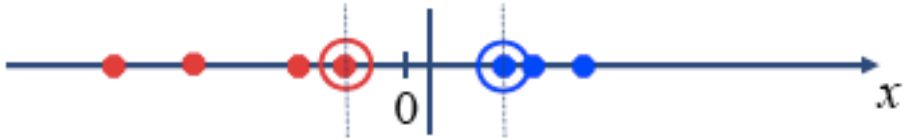
- Datasets that are linearly separable work out great:



- But what if the dataset is just too hard?

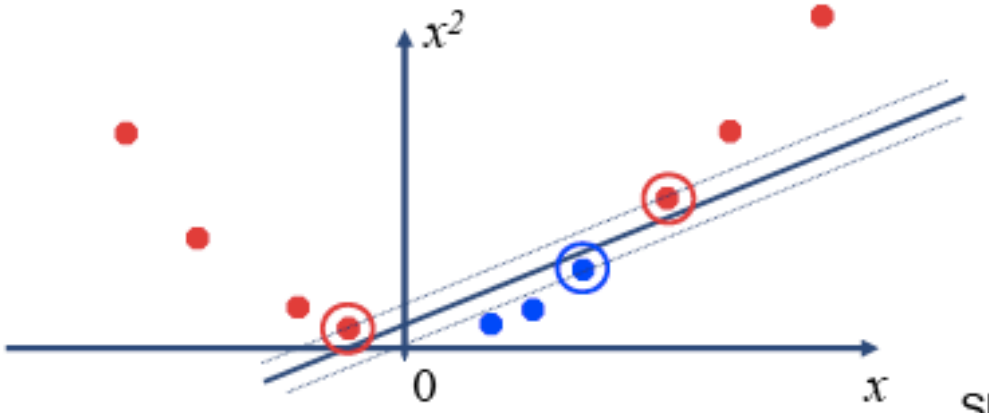
Classifier Using Labeled Data

- Datasets that are linearly separable work out great:



- But what if the dataset is just too hard?

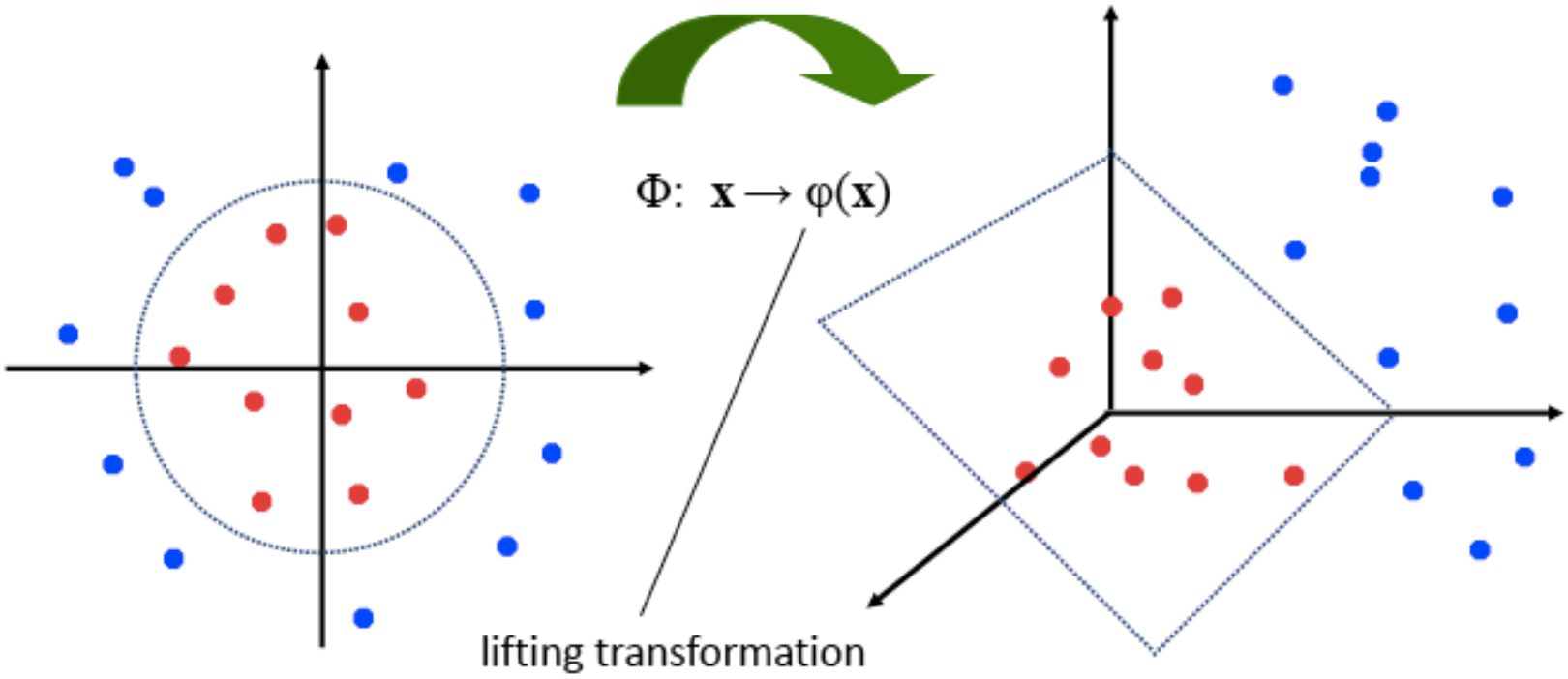
- We can map it to a higher-dimensional space:



Slide credit: Andrew Moore

"Kernel Trick"

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable:



Slide credit: Andrew Moore

A Classifier: Kernel Regularized Least Square Classifier

Given labeled training samples:

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)$$

- and a "kernel" $\kappa(\mathbf{x}, \mathbf{x}') = \langle \varphi(\mathbf{x}), \varphi(\mathbf{x}') \rangle$

(Non-linear) transformation

dot product

- Would like to find a classifier function f(z; α) that given a sample z, would return its label:

$$y = f(\mathbf{z}; \alpha) = \sum_{i=1}^{l} K(\mathbf{z}, \mathbf{x_i}) \alpha_i$$

the "distances" between z and all the labeled samples

A Classifier: Kernel Regularized Least Square Classifier

- Would like to find a classifier function f(x; α) that given a sample x, would return its label:

$$y = f(\mathbf{z}; \alpha) = \sum_{i=1}^{l} K(\mathbf{z}, \mathbf{x_i})\alpha_i$$

- Pose it as a LSE problem:

$$\min F(\alpha) = \min_{\alpha \in R^l} \frac{1}{2l} \sum_{i=1}^{l} (\mathbf{y} - K\alpha)^T (\mathbf{y} - K\alpha) + \frac{\lambda}{2} \alpha^T K \alpha$$

"good oracle" term

"regularization term" puts a bound on α

A Classifier: Kernel Regularized Least Square Classifier

$$\min F(\alpha) = \min_{\alpha \in R^l} \frac{1}{2l} \sum_{i=1}^{l} (\mathbf{y} - K\alpha)^T (\mathbf{y} - K\alpha) + \frac{\lambda}{2} \alpha^T K \alpha$$

$$\nabla_\alpha F(\alpha) = 0 = \frac{1}{l} \sum_{i=1}^{l} (-K\mathbf{y} + K^2\alpha) + \lambda K\alpha = -K\mathbf{y} + K^2\alpha + \lambda K\alpha$$

$$\boxed{\mathbf{y} = (K + \lambda I)\alpha}$$

Can solve for α☺

Need to invert a LARGE lx $l$matrix ☹

KRLSC

Learning:

Given a set of labeled samples, need to compute α

$$\mathbf{y} = (K + \lambda I)\alpha$$

Matrix Inversion

Testing:

Given an unlabeled sample z, need to compute y:

$$y = f(\mathbf{z}; \alpha) = \sum_{i=1}^{l} K(\mathbf{z}, \mathbf{x_i})\alpha_i = K(\mathbf{z}, \mathbf{x_i})^T \alpha$$

Distances computation, Cross-correlation

Efficiency considerations

We want:

- to use a large number of samples (dense sampling)
- the samples themselves are high dimensional (hundreds, thousands of pixels)
- to learn a new classifier at each frame
- test many unlabeled samples to find a “good” positive example

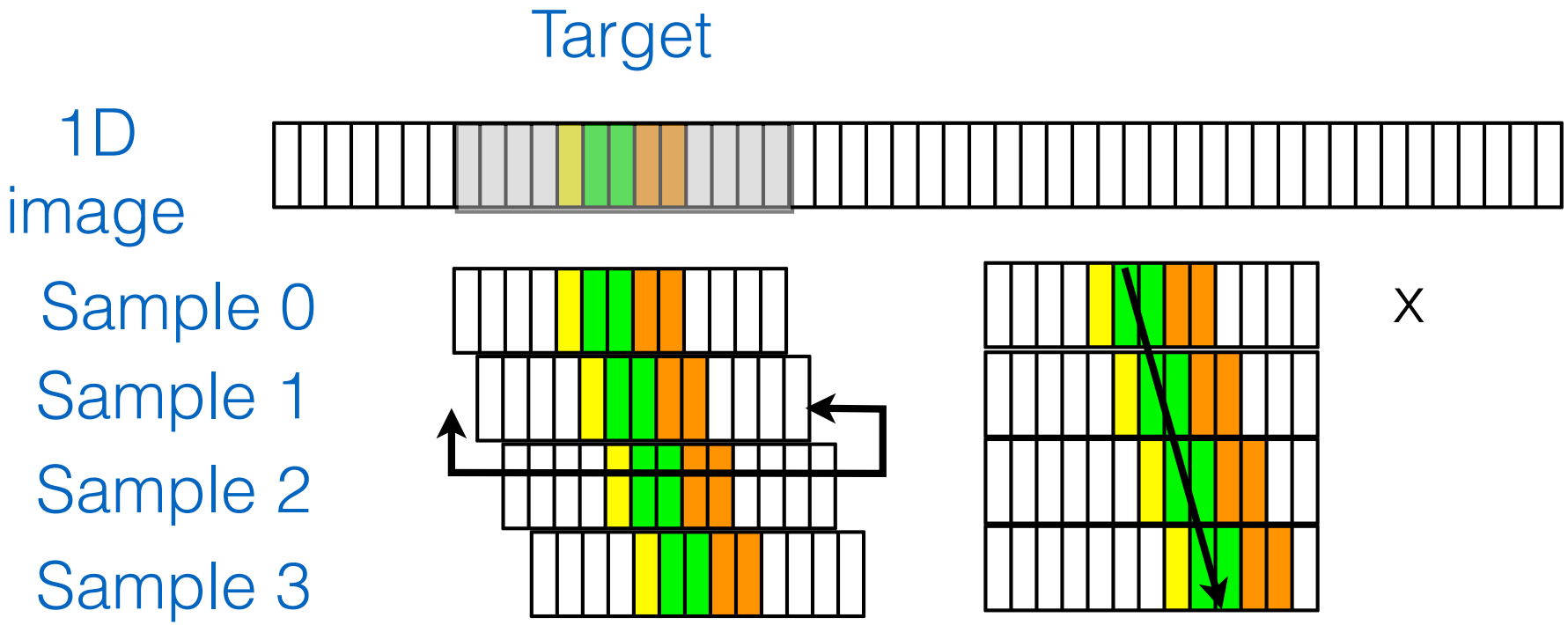
Solution:

- Use “circulant” matrices
- Make computations in the FREQUENCY domain
  - instead of multiplying matrices, we can multiply element by element
  - instead of inverting matrices, we can divide element by element

Price we pay:

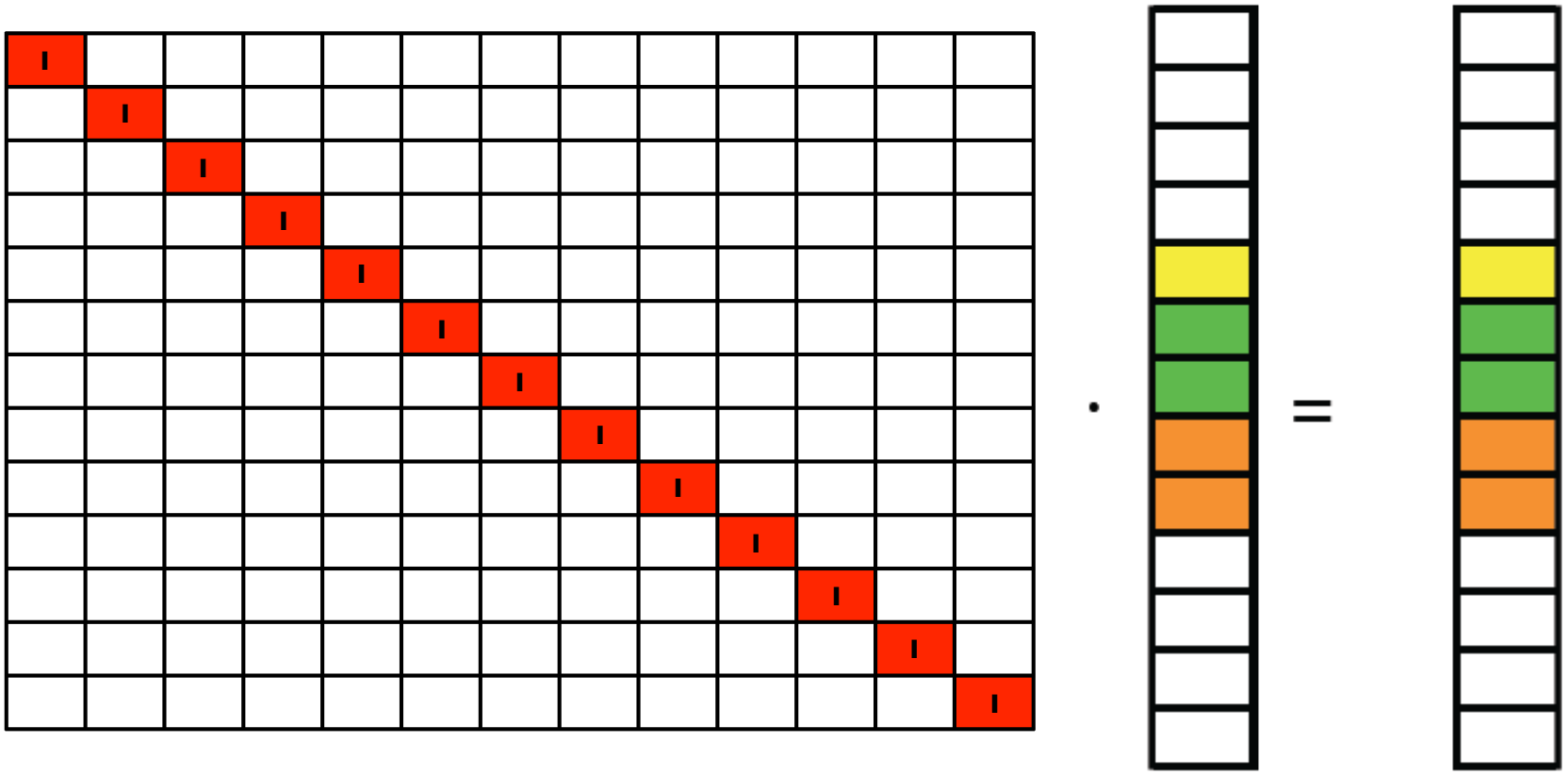
- We need to go to the Frequency domain and come back

“Dense Sampling” = Circulant Matrix



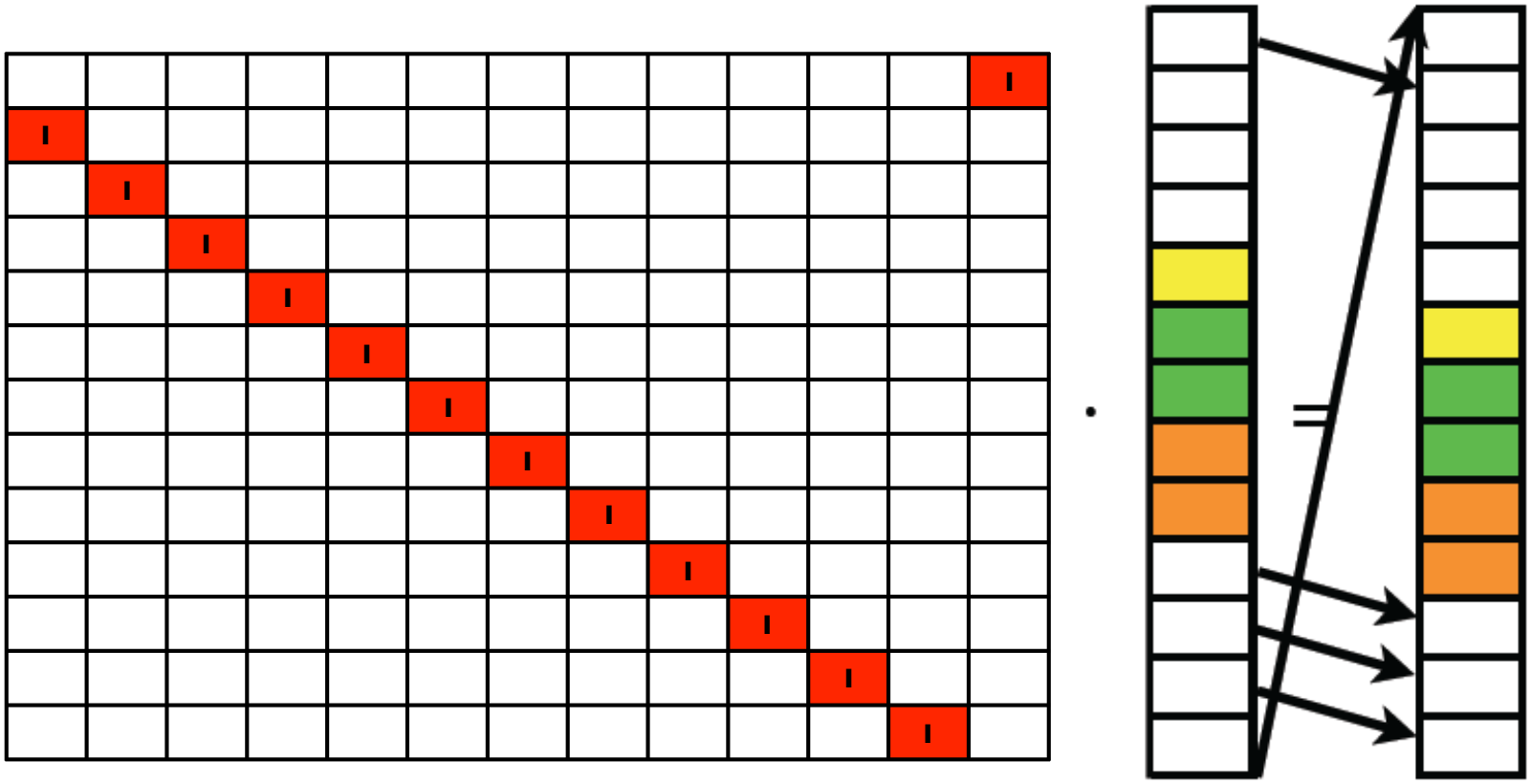
C(x) Circulant Matrix with constant diagonals

$$\mathbf{x_o} = I\mathbf{x} = P^0\mathbf{x_o}$$



Permutation Matrix: P

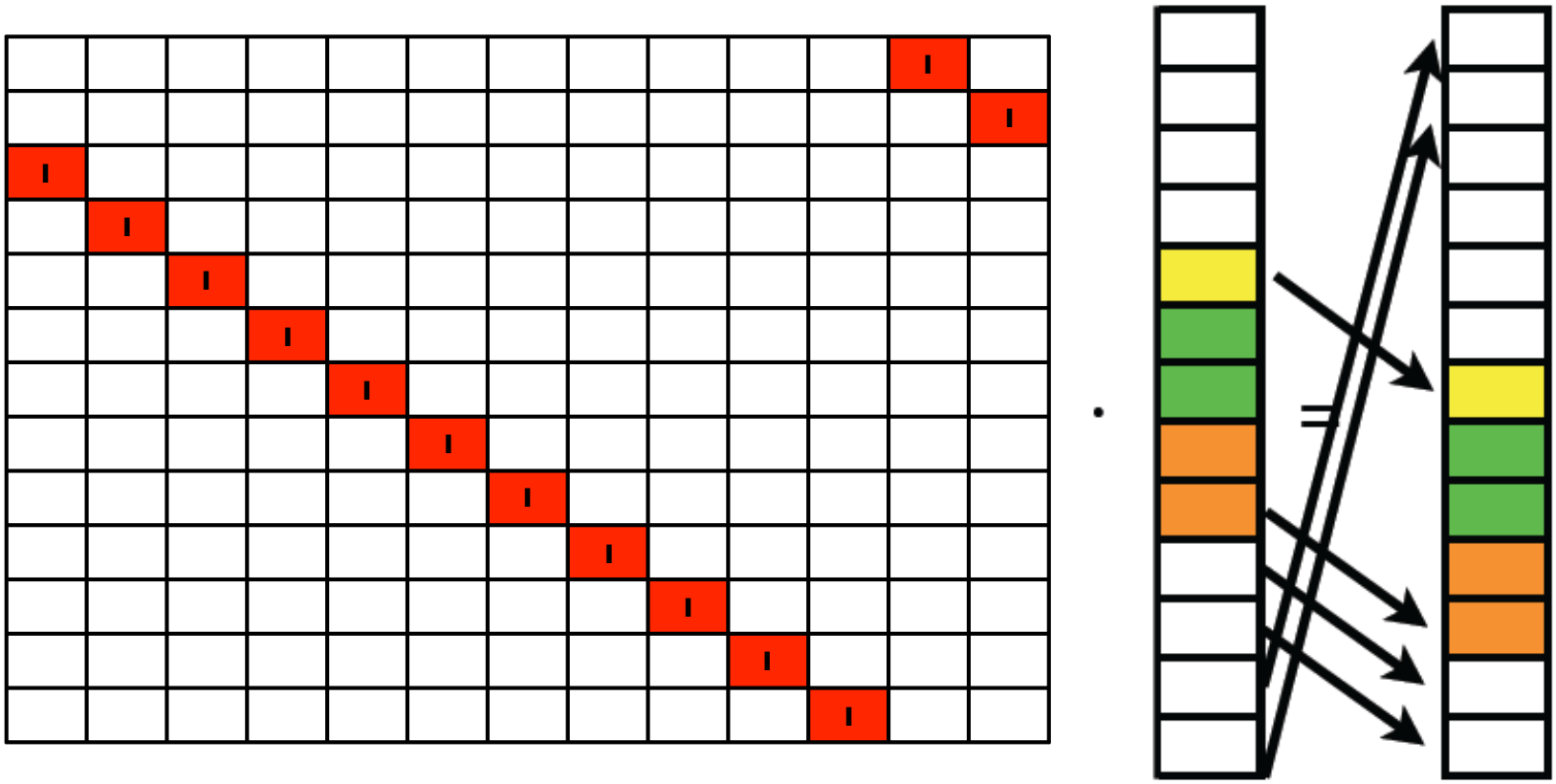
$$\mathbf{x_1} = P\mathbf{x}$$



Only one 1 in each row and in each column

Permutation Matrix: P

$$\mathbf{x_2} = P\mathbf{x_1} = PP\mathbf{x} = P^2\mathbf{x}$$



Only one 1 in each row and in each column

Kernels & Circulant Matrices

$$C(\mathbf{u}) = \begin{bmatrix} u_0 & u_1 & u_2 & \cdots & u_{n-1} \\ u_{n-1} & u_0 & u_1 & \cdots & u_{n-2} \\ u_{n-2} & u_{n-1} & u_0 & \cdots & u_{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ u_1 & u_2 & u_3 & \cdots & u_0 \end{bmatrix}$$

$$c_{ij} = u_{(j-i) \bmod n}$$

$$\mathbf{x}_i = P^i \mathbf{x}, \quad \forall i = 0, \ldots, n-1$$

n is the number of samples

Theorem 1. *The matrix K with elements $K_{ij} = \kappa(P^i\mathbf{x}, P^j\mathbf{x})$ is circulant if κ is a unitarily invariant kernel.*

Proof: $K_{ij} = \kappa(P^i\mathbf{x}, P^j\mathbf{x})$

κ is unitarily invariant if $\kappa(\mathbf{x}, \mathbf{x}') = \kappa(U\mathbf{x}, U\mathbf{x}')$ for any unitary matrix U

Permutations are unitary matrices.

$$K_{ij} = \kappa(P^{-i}P^i\mathbf{x}, P^{-i}P^j\mathbf{x}) = \kappa(\mathbf{x}, P^{j-i}\mathbf{x})$$

depends only on (j-i)mod n.

QED

The Frequency Domain

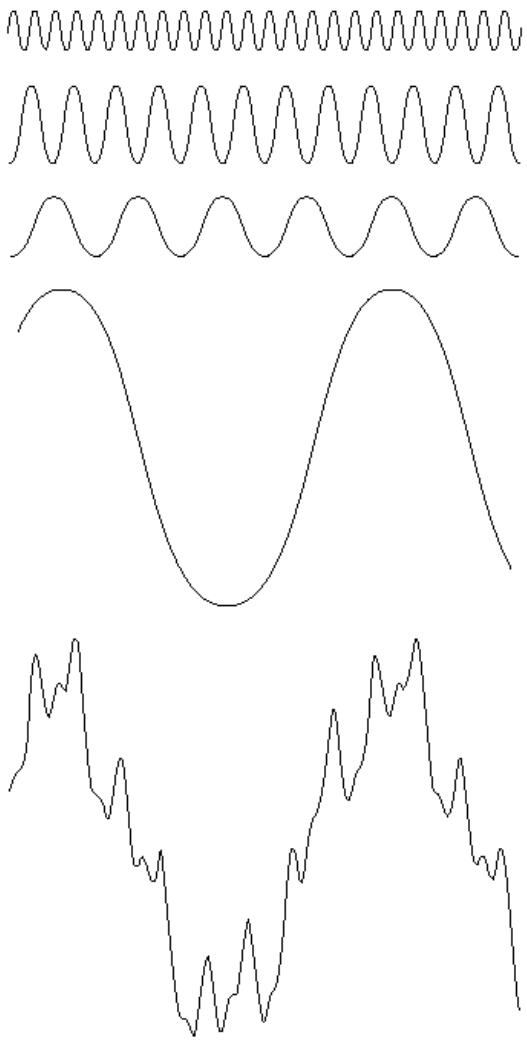


FIGURE 4.1 The function at the bottom is the sum of the four functions above it. Fourier's idea in 1807 that periodic functions could be represented as a weighted sum of sines and cosines was met with skepticism.

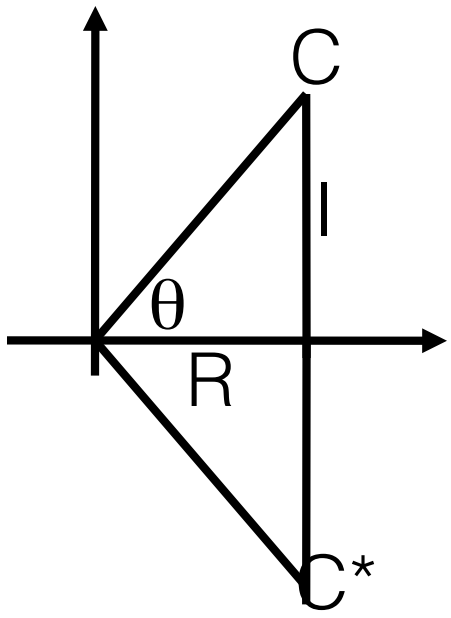
The sum of the four top functions (single frequencies) gives this "noisy" looking signal.

Review: Complex Numbers

Complex Numbers: $C = R + jI$

Conjugate: $C^* = R - jI$

Polar Coordinates: $C = |C|(\cos\theta + j\sin\theta)$



with $|C| = \sqrt{R^2 + I^2}$

$$\tan\theta = \frac{I}{R}$$

Using Euler's formula: $C = |C|e^{j\theta}$

$$e^{j\theta} = \cos\theta + j\sin\theta$$

Sampling f(x)

Samples of f(x) and F(u) are not necessarily at integer values, but they should be at equally spaced points:

$$f(x) \doteq f(x_o + k\Delta x)$$ With **k** integer

$$F(u) \doteq F(u\Delta u)$$ With **u** integer

Δx and **Δu** are related: $$\Delta u = \frac{1}{M\Delta x}$$

Discrete Fourier Transform (DFT)

One Dimension :

Let f(x) be a discrete function of x = 0,1,2,…,M-1

$$F(u) = \sum_{x=0}^{x=M-1} f(x) e^{-j2\pi ux/M} \quad u = 0, 1, \ldots, M-1$$

$$f(x) = \frac{1}{M} \sum_{u=0}^{u=M-1} F(u) e^{+j2\pi ux/M} \quad x = 0, 1, \ldots, M-1$$

NOTE: f(x) is real BUT F(u) will be, in general, complex

Euler's Formula and the Frequency Domain

$$F(u) = \sum_{x=0}^{x=M-1} f(x)e^{-j2\pi ux/M} \quad u = 0, 1, \ldots, M-1$$

Euler's formula: $e^{j\theta} = \cos\theta + j\sin\theta$

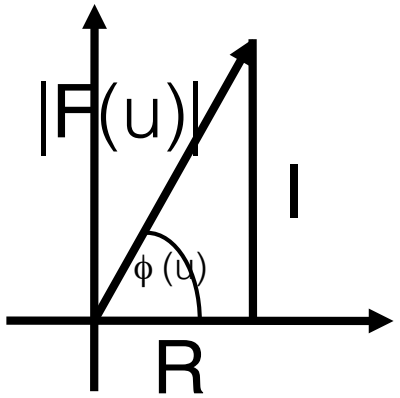
$$F(u) = \sum_{x=0}^{x=M-1} f(x)[\cos\frac{2\pi ux}{M} - j\sin\frac{2\pi ux}{M}] \quad u = 0, 1, \ldots, M-1$$

- For each value of u, we need to sum ALL values of f(x)
- u determines the frequency of the transform

Fourier Transform

$$F(u) = R(u) + jI(u)$$

$$F(u) = |F(u)|e^{j\phi(u)}$$



where $|F(u)| = \sqrt{R^2(u) + I^2(u)}$ "spectrum of f(x)"

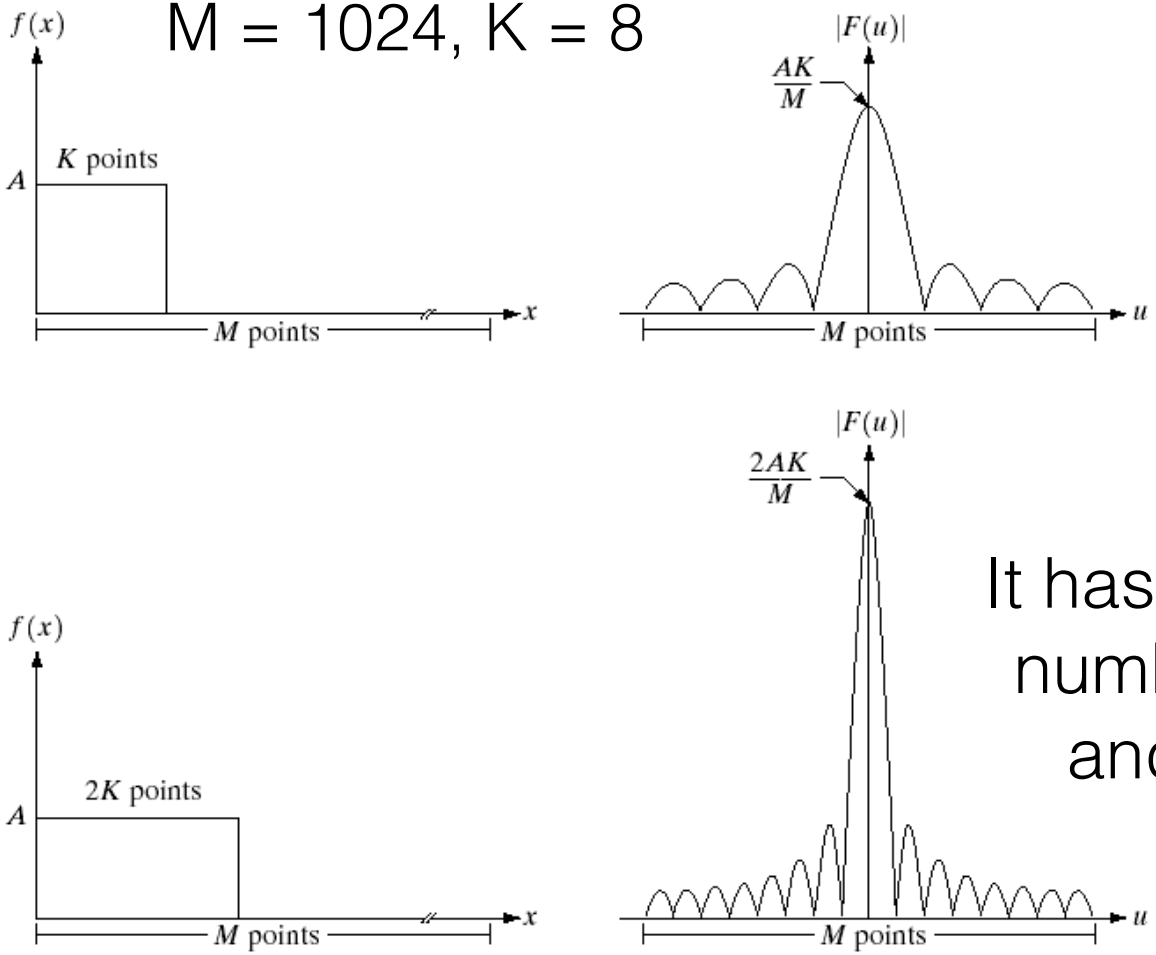
$P(u) = |F(u)|^2$ "power spectrum or spectral density of f(x)"

$\phi(u) = \tan^{-1}\frac{I(u)}{R(u)}$ "phase angle"

1D DFT Example

The functions are **discrete**, only drawn as continuous for visualization!

M = 1024, K = 8



a b
c d

FIGURE 4.2 (a) A discrete function of M points, and (b) its Fourier spectrum. (c) A discrete function with twice the number of nonzero points, and (d) its Fourier spectrum.

It has doubled the number of zeros and doubled height

FT and Convolution

$$\mathcal{F}[f(t) * h(t)] = F(u)H(u)$$
$$\mathcal{F}[f(t)h(t)] = F(u) * H(u)$$

$$\boxed{f(t) * h(t) = \mathcal{F}^{-1}\left[\mathcal{F}(u).\mathcal{H}(u)\right]}$$

Circulant Matrices, Kernels + FFT = Efficient!

Learning:

Given a set of labeled samples, need to compute α

$$\boxed{\mathbf{y} = (K + \lambda I)\alpha}$$

Matrix Inversion

$K = C(\mathbf{k})$ Circulant matrix with columns:

$$k_i = \kappa(\mathbf{x}, P^i\mathbf{x}), \quad \forall i = 0, \ldots, n-1$$

$$\boldsymbol{\alpha} = \mathcal{F}^{-1}\left(\frac{\mathcal{F}(\mathbf{y})}{\mathcal{F}(\mathbf{k}) + \lambda}\right), \text{element-wise division!}$$

For $n \times n$ images, the proposed algorithm has a complexity of only $\mathcal{O}(n^2 \log n)$

Circulant Matrices, Kernels + FFT = Efficient!

- Testing:
  - Given an unlabeled sample z, need to compute y:

$$y = f(\mathbf{z}; \alpha) = \sum_{i=1}^{l} K(\mathbf{z}, \mathbf{x_i})\alpha_i = K(\mathbf{z}, \mathbf{x_i})^T \alpha$$

Distances computation, Cross-correlation

$$K = C(\mathbf{k})$$

$$k_i = \kappa(\mathbf{x}, P^i\mathbf{x}), \quad \forall i = 0, \ldots, n-1$$

$$\bar{k}_i = \kappa(\mathbf{z}, P^i\mathbf{x})$$

$$\hat{\mathbf{y}} = \mathcal{F}^{-1}\left(\mathcal{F}(\bar{\mathbf{k}}) \odot \mathcal{F}(\boldsymbol{\alpha})\right)$$

element-wise multiplication!

For $n \times n$ images, the proposed algorithm has a complexity of only $\mathcal{O}(n^2 \log n)$

Fast Kernel Computation

Linear case: no Kernel Trick, just dot product

$$\kappa(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$$

$$\mathbf{w} = \mathcal{F}^{-1}\left(\frac{\mathcal{F}(\mathbf{x}) \odot \mathcal{F}^*(\mathbf{y})}{\mathcal{F}(\mathbf{x}) \odot \mathcal{F}^*(\mathbf{x}) + \lambda}\right)$$

Fast Kernel Computation

Gaussian Kernel

$$\mathbf{k}^{\text{gauss}} = \exp\left(-\frac{1}{\sigma^2}\left(\|\mathbf{x}\|^2 + \|\mathbf{x}'\|^2 - 2\mathcal{F}^{-1}\left(\mathcal{F}(\mathbf{x}) \odot \mathcal{F}^*(\mathbf{x}')\right)\right)\right)$$

Implementation Details

Images are not really periodic ... bound them using a sinusoidal window:

$$\left(x_{ij}^{\text{raw}} - 0.5\right) \sin(\pi i/n) \sin(\pi j/n), \quad \forall i, j = 0, \ldots, n-1$$

- Use a “continuous” class label to yield smooth spatial responses:

$$y_{ij} = \exp\left(-\left((i-i')^2 + (j-j')^2\right)/s^2\right), \quad \forall i, j = 0, \ldots, n-1$$

- “Incorporates temporal memory” by linearly interpolating the new parameters with the ones from the previous frame:

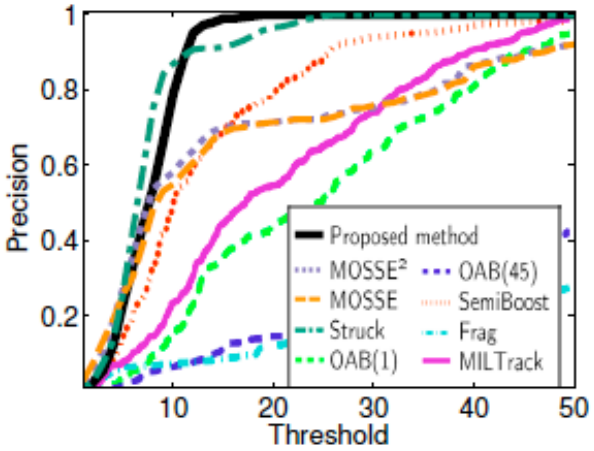
```
alphaf = (1 - interp_factor) * alphaf + interp_factor * new_alphaf;
       z = (1 - interp_factor) * z + interp_factor * new_z;
```

CODE

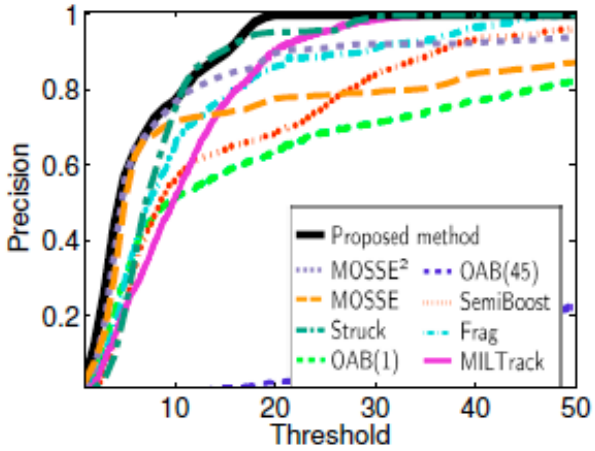
Algorithm 1 . MATLAB code for our tracker, using a Gaussian kernel
It is possible to reuse some values, reducing the number of FFT calls. An implementation with GUI is available at: `http://www.isr.uc.pt/~henriques/`

```
% Training image x (current frame) and test image z (next frame)
% must be pre-processed with a cosine window. y has a Gaussian
% shape centered on the target. x, y and z are M-by-N matrices.
% All FFT operations are standard in MATLAB.

function alphaf = training(x, y, sigma, lambda)        % Eq. 7
  k = dgk(x, x, sigma);
  alphaf = fft2(y) ./ (fft2(k) + lambda);
end

function responses = detection(alphaf, x, z, sigma)   % Eq. 9
  k = dgk(z, x, sigma);
  responses = real(ifft2(alphaf .* fft2(k)));
end

function k = dgk(x1, x2, sigma)                        % Eq. 16
  c = fftshift(ifft2(fft2(x1) .* conj(fft2(x2))));
  d = x1(:)'*x1(:) + x2(:)'*x2(:) - 2*c;
  k = exp(-1 / sigma^2 * abs(d) / numel(x1));
end
```

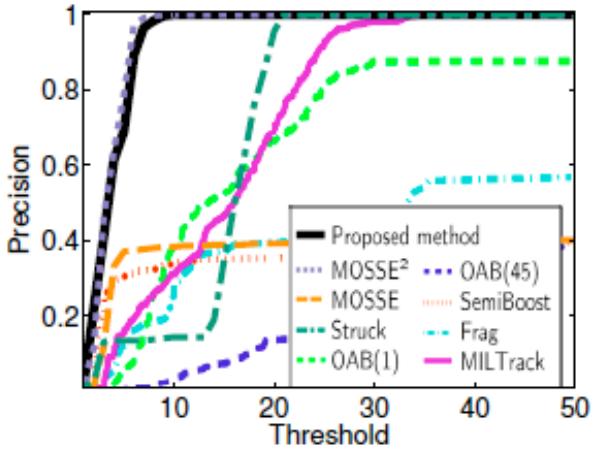
Performance (300 fps)



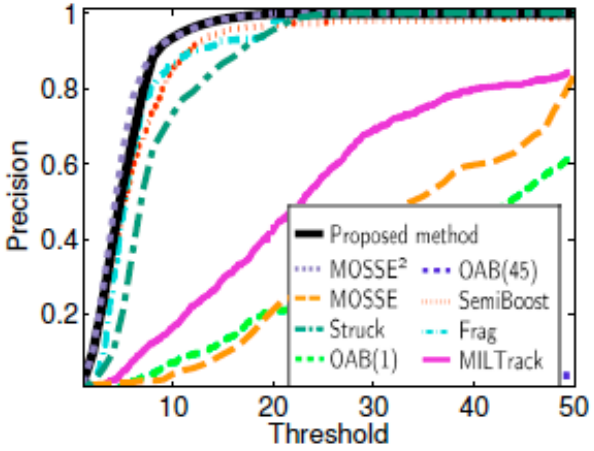
(a) coke11



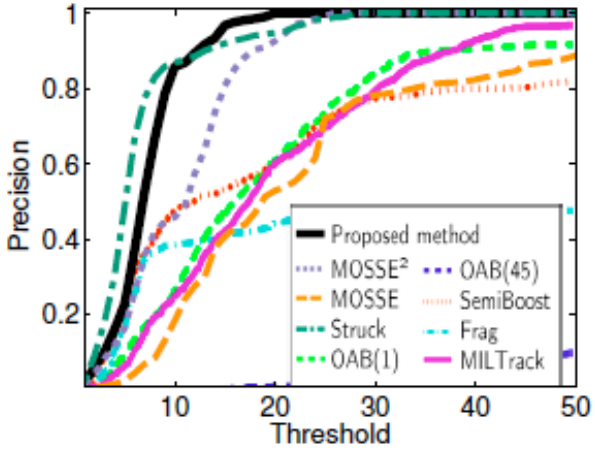
(b) sylvester



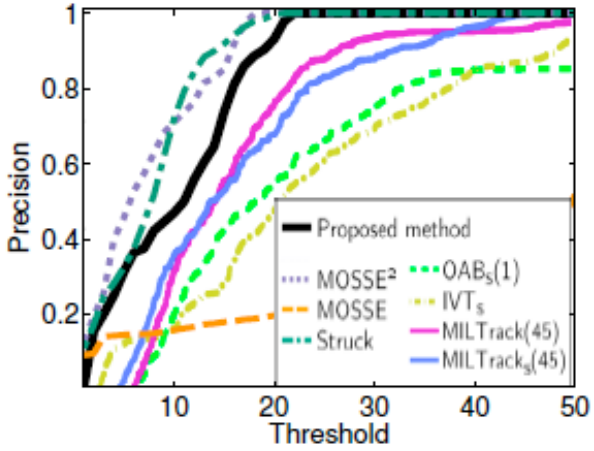
(c) dollar



(d) faceocc

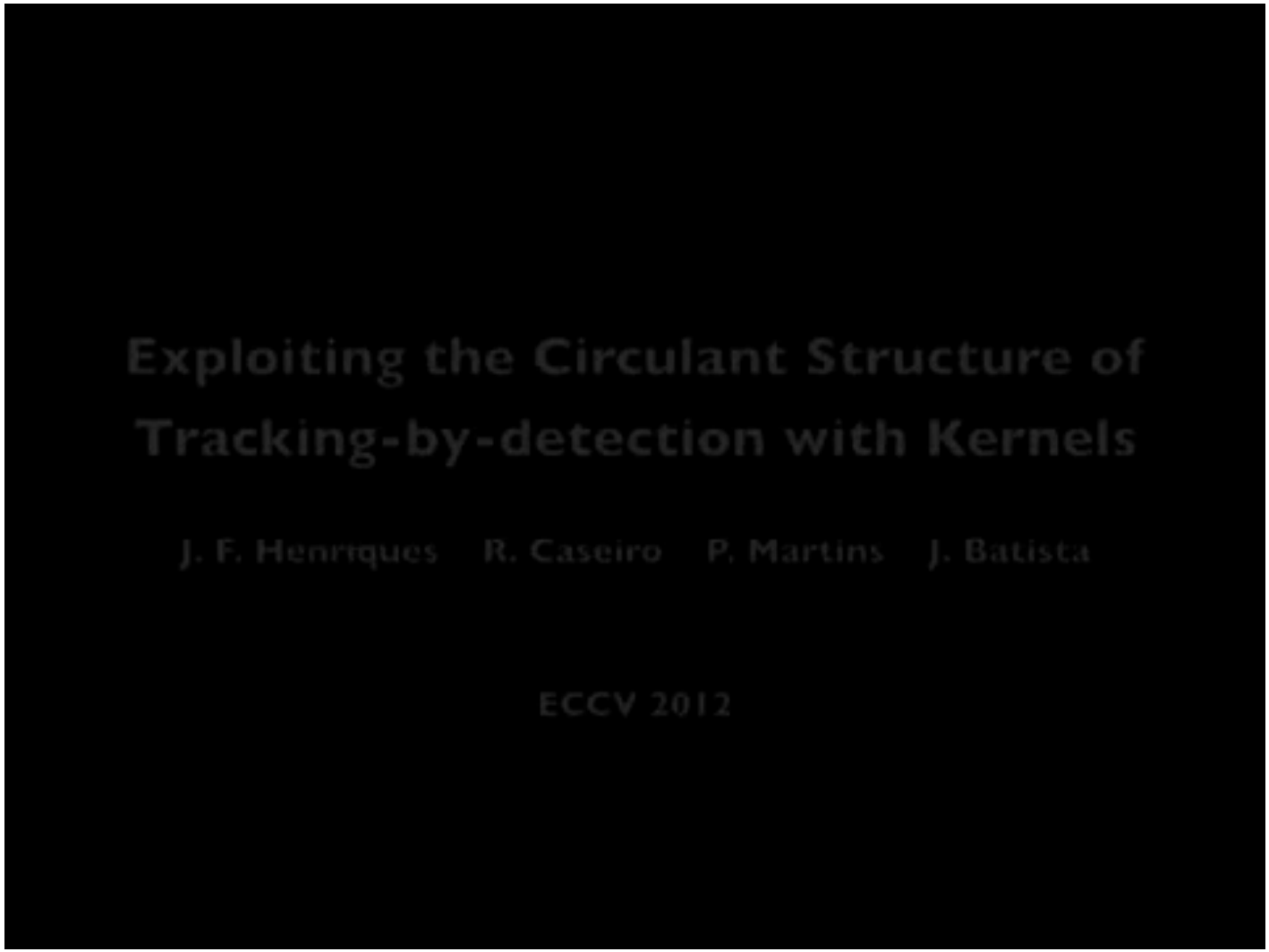


(e) faceocc2



(f) twinings

Videos



http://www.youtube.com/watch?v=sZXKnPUhAi8